Návrh na vytvoření REST / SOAP služby pre přístup do databáze

Zita Bajúszová

Diplomová práce
2017

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2016/2017

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Mgr. Zita Bajúszová**
Osobní číslo:	**A15216**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**kombinovaná**
Téma práce:	**Návrh a vytvoření REST / SOAP služby pro přístup do databáze**
Téma anglicky:	**The Design and Creation of REST / SOAP Services for Access to a Database**

Zásady pro vypracování:

1. **Stanovte cíl projektu – business model, seznam funkčních a nefunkčních požadavků.**
2. **Prostudujte a popište vhodné softwarové prostředky pro dosažení stanovených cílů.**
3. **Definujte základní případy použití pro navrhovaný informační systém.**
4. **Vytvořte logický datový model datového skladu a navrhněte databázi.**
5. **Navrhněte wireframe pro frontend a popište rozhraní pro backendové služby.**
6. **Uvedený návrh realizujte a tuto realizaci popište.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **RICHARDSON, Leonard a Michael AMUNDSEN. RESTful Web APIs. O'Reilly Media, 2013. ISBN 9781449358068.**
2. **ERL, Thomas: SOA Servisně orientovaná architektura, Computer Press, 2009. ISBN: 9788025118863**
3. **Apache Maven Project [online]. [cit. 2017-02-01]. Dostupné z: https://maven.apache.org/**
4. **GIT - free and open source distributed version control system [online]. [cit. 2017-02-01]. Dostupné z: https://git-scm.com**
5. **PlantText- The expert's design tool [online]. [cit. 2017-02-01]. Dostupné z: https://www.planttext.com**

Vedoucí diplomové práce: **doc. Ing. Jiří Vojtěšek, Ph.D.**
Ústav řízení procesů

Datum zadání diplomové práce: **3. února 2017**

Termín odevzdání diplomové práce: **16. května 2017**

Ve Zlíně dne 3. února 2017

L.S.

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne …….

podpis diplomanta

ABSTRAKT

Cieľom diplomovej práce je navrhnúť RESTové API pre prístup do databázy. Samotnému návrhu predchádza štúdium teoretických informácií o RESTovom rozhraní, jeho bezpečnosti a s ním súvisiacom HTTP protokole. Na základe týchto informácií je vytvorené požadované API a jeho funkčnosť následne demonštrovaná v rámci jednoduchej aplikácie.

Okrem návrhu samotného API je v štádiu prípravy stanovený zoznam funkčných a nefunkčných požiadaviek, kladených na aplikáciu. Následne sú zvolené softvérové prostriedky ako pre frontend, tak pre backend. Vybrané prostriedky popisuje samostatná kapitola. Návrh aplikácie ďalej obsahuje návrh databázy, wireframu pre fronted a popis rozhraní pre backendové služby.

V poslednej fáze nasleduje realizácia navrhnutého API aj spolu s aplikáciou.

Klíčová slova: REST, API, HTTP, webové aplikácie, Angular, Node.js, Express

ABSTRACT

The aim of this thesis is to design REST API for access to the database. The design itself is preceded by a study of the theoretical information about the REST interface, its security and HTTP protocol. Based on this information, the required API is created and its functionality is demonstrated in a simple application.

In addition to the API itself, a list of functional and non-functional application requirements is set. Then software tools are selected for both frontend and backend. The selected tools are describe in a separate section. The application design includes database design, wireframe for fronted, and interface description for backend services as well.

In the last phase, the proposed API is implemented together with the application

Keywords: REST, API, HTTP, web application, Angular, Node.js, Express

Rada by som poďakovala svojmu vedúcemu diplomovej práce doc. Ing. Jiřímu Vojtěškovi, Ph.D. a konzultantovi Ing. Jozefovi Burešovi z firmy MONET+, a.s. za odborné vedenie, podnetné rady, informácie a trpezlivosť, ktoré mi poskytovali počas práce na mojej diplomovej práci.

OBSAH

ÚVOD

Motivácia

S príchodom Internetu sa markantne zmenil spôsob, ako zdieľame informácie. Internet je teraz už takmer všade, ľudia sú online doslova 24 hodín denne a informácie akéhokoľvek druhu hľadajú primárne na webe. S rozmachom Internetu sa začal aj prudký vývoj webových aplikácii. Ich výhodou je, že na ich použitie stačí internetový prehliadač, nevyžadujú žiaden drahý hardware a informácie sa môžu zdielať medzi sebou po sieti, čo uľahčuje programátorom prácu. Každá webová aplikácia je totiž akousi webovou službou – programom spusteným na serveri a prostredníctvom web API (aplikačné rozhranie) môže komunikovať s ďalšou webovou službou, získavať od nej dáta, alebo meniť jej stav. Aplikácia zároveň môže prostredníctvom API získavať dáta od užívateľov a ukladať si ich do databázy.

Moja práca sa zaoberá návrhom a implementáciou takéhoto webového aplikačného rozhrania, postaveného na RESTovej architektúre.

Cieľ práce

Cieľom diplomovej práce je

navrhnúť RESTové API pre prístup do databázy. Samotnému návrhu predchádzalo štúdium teoretických informácií o RESTovom rozhraní, jeho bezpečnosti a s ním súvisiacom HTTP protokole. Na základe týchto informácií bolo vytvorené požadované API a jeho funkčnosť následne demonštrovaná v rámci jednoduchej aplikácie.

Okrem návrhu samotného API bol v štádiu prípravy stanovený zoznam funkčných a nefunkčných požiadaviek, kladených na aplikáciu. Následne boli zvolené softvérové prostriedky ako pre frontend, tak pre backend. Vybrané prostriedky popisuje samostatná kapitola. Návrh aplikácie ďalej obsahuje návrh databázy, wireframu pre fronted a popis rozhraní pre backendové služby.

Stručné predstavenie RESTu

REST je skratka pre Representational State Transfer. Jedná sa o architektonický štýl, ktorý vo svojej dizertačnej práci predstavil Roy Thomas Fielding. Nie je závislý na jazyku, protokole, ani platforme. Jeho hlavnou charakteristikou je oddelená implementácia servera od klienta. Medzi základné princípy RESTu patrí:

- každý zdroj musí mať svoj vlastný identifikátor (napr. URL);
- stav aplikácie sa určuje pomocou URL;
- pre získanie a manipuláciu so zdrojmi existuje jednotný prístup, ktorý zahŕňa štyri základné operácie CRUD (viac viď kapitola 2. Restové služby);
- zdroj môže mať rôzne reprezentácie (napr. XML, JSON, SVG), klient následne pracuje s touto reprezentáciou, nie priamo so zdrojom.

REST teda nemôžeme chápať ako návod na vytvorenie dizajnu alebo implementáciu aplikácie. Jedná sa „iba“ o sadu podmienok a obmedzení, ktoré musí aplikácia spĺňať, aby bola označená ako RESTová.

Prehľad členenia diplomovej práce

Diplomová práca je členená na dve časti a to Teoretickú a Praktickú. Teoretická časť sa zameriava na hlbšiu analýzu danej problematiky, Praktická časť popisuje spôsob návrhu a realizácie API a aplikácie.

Teoretickú časť tvoria tri kapitoly, konkrétne 1. Základné pojmy, 2. RESTová architektúra, a 3. Analýza frameworkov pre tvorbu RESTových rozhraní. Prvá kapitola, ako naznačuje už jej názov, predstavuje základné pojmy, ktoré súvisia s problematikou komunikácie webových aplikácií po sieti. Druhá kapitola sa venuje princípom RESTovej architektúry, popisom nefunkčných požiadaviek na RESTové rozhranie a ďalším témam, súvisiacim aplikáciou RESTovej architektúry. Tretia kapitola obsahuje podrobnejší opis najčastejšie využívaných frameworkov, používaných na vývoj API.

Praktickú časť tvoria tak isto tri kapitoly, a to 4. Analýza požiadaviek a 5. Použité technológie a 6. Návrh aplikácie a realizácia.

Výstupom je funkčná aplikácia, ktorá tvorí prílohu diplomovej práce.

I. TEORETICKÁ ČÁST

1 ZÁKLADNÉ POJMY

Pre lepšie pochopenie RESTových služieb je potrebné, aspoň stručne, vysvetliť niektoré základné pojmy, ktoré objasnia miesto týchto služieb v širšej tématike webových aplikácií. Týmto základným pojmom, resp. oblastiam, sa bude venovať nasledujúca kapitola.

Jedná sa o:

- Vysvetlenie pojmu webová aplikácia;
- Stručné zhrnutie webových komponentov a ich funkcionalít, ktoré vstupujú medzi odosielateľa a adresáta http(s) správy, a ktoré ovplyvňujú bezpečnosť a rýchlosť komunikácie;
- Úvod HTTP(s) protokolu. Problematike komunikácie prostredníctvom tohto protokolu sa venujem podrobnejsič v kapitole 2. RESTové služby. Jedná sa ale o základný pojem, preto ho stručne uvádzam aj medzi základnými pojmami v širšom kontexte webových aplikácií;
- Objasnenie základných princípov a histórie webových služieb.

1.1 Webová aplikácia

Webová aplikácia je aplikácia, ktorá je umiestnená na webovom serveri, odkiaľ je dostupná užívateľom prostredníctvom počítačovej siete. Takéto aplikácie sú, na rozdiel od desktopových aplikácií, schopné vzájomnej interakcie prostredníctvom webových služieb.

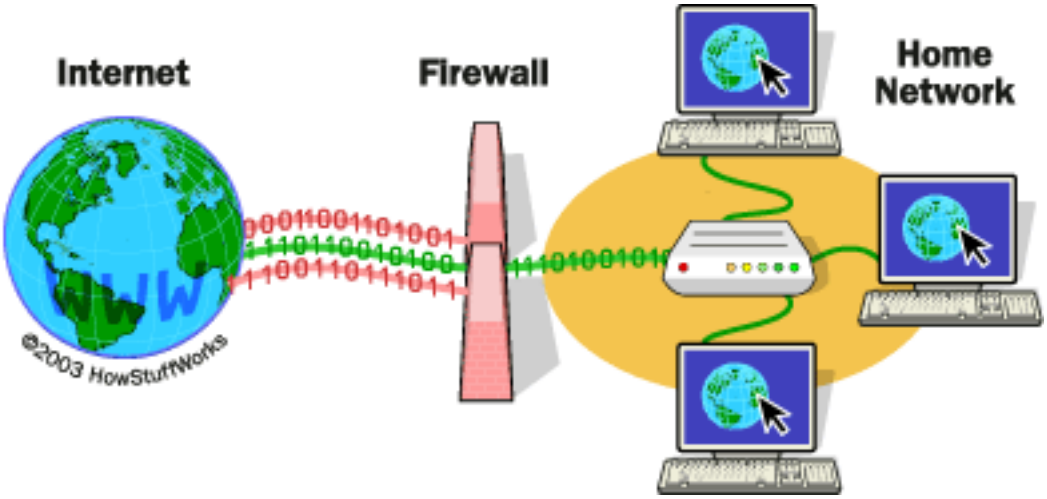
1.2 Webové komponenty

Webové komponenty sú dôležitou súčasťou webových aplikácií:

- Firewally – zabezpečujú webovú bezpečnosť;
- Routre – zabezpečujú webovú škálovateľnosť;
- Caches – zabezpečujú webovú rýchlosť.

1.2.1 Firewall

Firewall je sieťové zariadenie, ktoré riadi a zabezpečuje prevádzku medzi sieťami s rôznym stupňom zabepečenia. Zjednodušene povedané, definujú pravidlá, ktoré rozhodujú, ktoré HTTP(s) správy (messages) môžu dnu, a ktoré von.



Obrázok 1: Príklad implementácie Firewallu – kontrola prevádzky medzi domácou sieťou a Internetom.[1]

Tieto pravidlá v minulosti zahrňovali zdrojovú a cieľovú IP adresu a port. V súčasnosti to už nepostačuje, firewally aktuálne spracúvajú aj ďalšie informácie, ako napríklad informácie o stave spojenia, kontrolu známych protokolov a prvky IDS (systém pre odhalenie podozrivých aktivít).

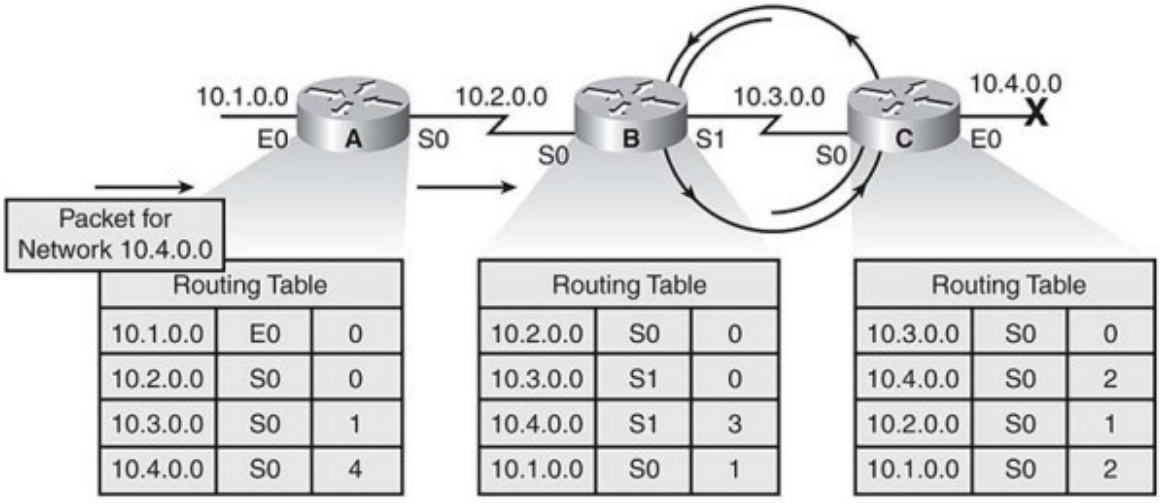
Znamená to, že firewally dokážu kontrolovať spojenie až na úroveň dát protokolov a aplikácií a zakázať napríklad priechod samotného HTTP spojenia pokiaľ usúdia, že sa nejedná o požiadavku na daný www server.

1.2.2 Router

Router je ďalším zo sieťových zariadení, ktoré nám pomáha pri využívaní webových aplikácií. Jeho úlohou je preposielanie pakety smerom od odosielateľa k adresátovi. Tzn., že routre rozhodujú o tom, kam poslať http správy. Tento proces sa označuje ako smerovanie (routovanie). Každé smerovacie zariadenie obsahujú sadu pravidiel, ktoré určujú, kam zariadenie má, resp. môže pakety poslať. Za týmto účelom sa využíva smerovacia tabuľka, ktorá obsahuje informácie nutné pre rozhodovanie pri smerovaní. Každý záznam obsahuje:

[1] Zdroj: http://s.hswstatic.com/gif/firewall.gif [cit. 2017-04-16].

- Cieľ – číslo cieľovej podsiete vo forme IP adresy;
- Maska podsiete – používa sa pre určenie adresy podsiete;
- Brána – IP adresa routra, kam paket smeruje ďalej;
- Sieťové rozhranie.



Obrázok 2: Smerovacia tabuľka[2]

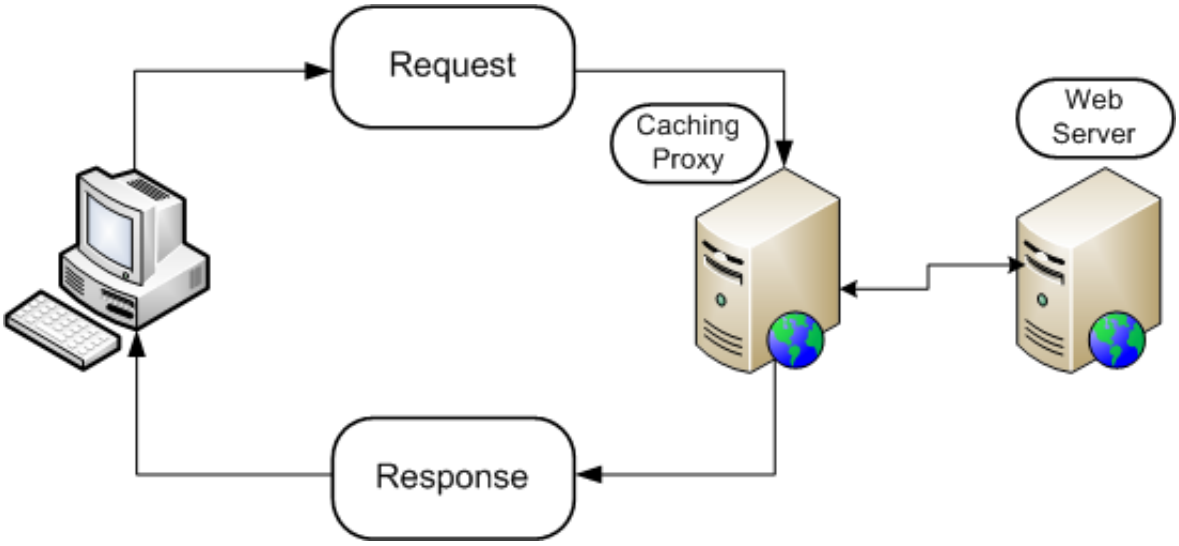
1.2.3 Cache

Cache medzipamäť je súčasť počítača, alebo servera, ktorá uchováva dáta, a tým urýchľuje proces prístupu k týmto dátam. Cache môže byť softvérová alebo hardvérová. Webové aplikácie využívajú najmä webovú cashe a cache mapovanie doménových mien a IP adresy, ktorú poskytuje DNS server.

Webová cache ukladá predchádzajúce odpovede od webových serverov a znova ich využíva, čím znižuje záťaž siete a webových serverov. Tzn. že cache sa rozhoduje, či použiť nacachovanú kópiu HTTP dokumentu, alebo si vyžiadať aktuálnejšiu verziu z webového servera. Rozhodnutia sa dejú na základe HTTP hlavičky. Cache nepozerá do do prenášaných dát.

[2] Zdroj: https://duongtuanan.files.wordpress.com/2015/09/092415_0321_ccnarouting5.jpg?w=604 [cit. 2017-04-16].

Obvykle je súčasťou webových priehliadačov. Pre tento účel sa tiež používa proxy server, alebo reverzný proxy server, ktoré cachujú dáta smerujúce z webových serverov ku klientom.



Obrázok 3: Cachovanie pomocou proxy servera[3]

1.3 HTTP(s) protokol

HTTP protokol je internetový protokol, ktorý je určený na výmenu hypertextových dokumentov vo formáte HTML a pracuje na aplikačnej vrstve podľa TCP/IP modelu. Funguje na princípe request – response medzi klientom a serverom – klient pošle na server request vo forme čistého textu, ten požiadavku spracuje a pošle odpoveď naspäť. V rámci protokolu sú uvedené presné pravidlá, ako má komunikácia vyzerať. V súčasnosti sa využíva verzia HTTP 2.0.

K zabezpečeniu HTTP pripojenia sa používa HTTP(s) – syntakticky je rovnaké ako HTTP, len sa pre prenos dát používa šifrovacia metóda SSL/TLS.

Zjednodušená klientská požiadavka sa skladá z HTTP metódy, obsahu hlavičky, prázdneho riadka a samotných dát.

[3] Zdroj: http://www.ilian.io/wp-content/uploads/2011/05/Drawing21.png [cit. 2017-04-16].

Podrobnejšie sa HTTP(s) protokolu budem venovať v 2. kapitole, konkrétne v podkapitole, v ktorej predstavím komunikáciu prostredníctvom RESTových služieb.

1.4 Webové služby

Webovú službu (web service) (WS) môžeme definovať ako funkcionalitu jednej aplikácie, ktorú využíva iná aplikácia, za účelom ich vzájomnej interakcie. Jedná sa o strojovú komunikáciu, tzn. že táto komunikácia je napísaná v strojovo spracovateľnom formáte. Bežne priebeha napríklad pomocou posielania správ vo formáte XML a protokolu HTTP(s).

Existuje niekoľko spôsobov a rôznych prístupov ako implementovať WS. Tri najčastejšie spôsoby použitia sú:

- RPC;
- SOA;
- REST.

1.4.1 RPC

Vzdialené volanie procedúr (RPC – Remote Procedure Call) je jednou z najstarších metód pre komunikáciu aplikácií na diaľku. Táto metóda poskytuje možnosť, ako volať funkciu, ktorá sa nachádza na inom mieste, ako volajúci program.

RPC je jednoduchý model, ktorý podporuje len základné dátové typy. Prenos štruktúrovaných dát, akými sú napríklad fragmenty XML, nie je priamo podporovaný.

Neskôr vznikol pre potreby RPC XML-RPC protokol, ktorý umožňoval zapuzdriť dáta pomocou XML a prenášať ich prostredníctvom HTTP protokolu. Tento protokol sa stal predlohou pre SOAP. Aj v súčasnej dobe existujú vývojari, ktorí preferujú XML-RPC pre jeho minimalizmus a jednoduchosť.

Príklady[4]

Typický RPC-XML request:

```
<?xml version="1.0"?>
<methodCall>
```

[4] Zdroj uvedených príkladov: https://en.wikipedia.org/wiki/XML-RPC [cit. 2017-04-16].

```
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
        <value><i4>40</i4></value>
    </param>
  </params>
</methodCall>
```

Typický RPC-XML response:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
        <value><string>South Dakota</string></value>
    </param>
  </params>
</methodResponse>
```

1.4.2 SOA

SOA znamená architektúru orientovanú na služby. V praxi to znamená, že pri správnom návrhu je možné pripojiť, odpojiť alebo nahradiť akúkoľvek službu bez toho, aby sa zasahovalo do aplikácie samotnej za predpokladu, že klient dopredu vie, akým spôsobom sa môže k službe pripojiť. Hranice systému sú pri takomto prístupe definované prostrednítvom špecifického formátu dát, pomocou ktorých daná služba komunikuje.

V súčasnosti sa väčšina dát prenáša prostredníctvom formátu XML. Komunikácia sa teda riadi SOA protokolom (SOAP - Simple Object Access Protocol) postavenom na XML, ktorý pre sieťovú komunikáciu využíva ďalšie protokoly aplikačnej vrstvy, najčastejsie HTTP, ktorú akceptuje väčšina firewallov.

Na rozdiel od RPC sa tu jednotkou komunikácie stáva správa (message), táto architektúra sa preto zvykne nazývať aj „message oriented“.

SOAP správa sa skladá z troch častí:

- Obálka (envelope) – obsahuje definíciu menných priestorov (namespaces), ktoré umožňujú rozlíšiť elementy popisujúce štruktúru SOAP od samotnej právy;
- Hlavička (header) – definuje operácie, ktoré je nutné vykonať pred spracovaním tela správy, napr. overenie práv a pod. Pokiaľ server týmto transakčným elementom nerozumie, je správa zamietnutá;
- Telo (body) – obsahuje informácie, na základe ktorých sa vykoná niektorá z požadovaných metód.

Na zjednodušenie použitia SOAPu vznikol neskôr štandard WSDL (Web Services Description Language), ktorý rozširuje možnosti tohto protokolu.

Konzorcium W3C, ktoré sa stará o WSDL, ho definuje nasledovne: „WSDL je XML formát pre opis sieťových služieb ako súboru koncových bodov, využívajúci správy, ktoré obsahujú buď dokumentovo alebo procedurálne orientované informácie. Operácie a správy sú popísané abstraktne, a až následne viazané na konkrétny sieťový protokol a formát správy pre definovanie koncového bodu.“ [1]

Príklady[5]

Typický SOAP request

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>
</soap:Envelope>
```

Typický SOAP response:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>
</soap:Envelope>
```

1.4.3 REST

REST (Representational State Transfer) kladie na rozdiel od XML – RPC alebo SOAP dôraz na interakciu so stavovými zdrojmi, nie so správami alebo procedúrami. Je teda

[5] Zdroj uvedených príkladov: https://www.w3schools.com/xml/xml_soap.asp [cit. 2017-04-16].

orientovaná dátovo. Rozdiel je tiež v tom, že sa nejedná o protokol, ale o architektúru rozhrania.

Medzi základné princípy RESTu patrí:

- každý zdroj musí mať svoj vlastný identifikátor (napr. URL);
- stav aplikácie sa určuje pomocou URL;
- pre získanie a manipuláciu so zdrojmi existuje jednotný prístup, ktorý zahŕňa štyri základné operácie CRUD (viac viď kapitola 2. Restové služby);
- zdroj môže mať rôzne reprezentácie (napr. XML, JSON, SVG), klient následne pracuje s touto reprezentáciou, nie pramo so zdrojom.

Zdroj	GET	PUT	POST	DELETE
URI kolekcia, napríklad http://example.com/resources/	**Zoznam** (List) URI a prípadne ďalší detaily členov kolekcie.	**Vymeniť** (Replace) celú kolekciu za inú.	**Vytvoriť** (Create) nový záznam do kolekcie. Jeho ID je automaticky pridelené a väčšinou vrátené touto operáciou.	**Zmazať** (Delete) celú kolekciu.
URI prvku, napríklad http://example.com/resources/142	**Vrátiť** (Retrieve) reprezentáciu adresovaného člena v kolekcii, vyjadreného vhodným internetovým typom média.	**Upraviť** (Update) adresovaný člen kolekcie, alebo – pokiaľ neexistuje – **vytvoriť** (create) ho.	Jednať s adresovaným členom ako s kolekciou a **vytvoriť** v ňom novú položku.	**Zmazať** (Delete) adresovaný prvok z kolekcie.

Tabuľka 1: Metódy HTTP pre webové služby, ktoré sú RESTful[6]

[6] Zdroj: https://cs.wikipedia.org/wiki/Representational_State_Transfer [cit. 2017-04-16].

1.4.4 Porovnanie SOAPu a RESTu

#	SOAP	REST
1	Protokol založený na posielaní správ vo forme XML	Architektúra rozhrania
2	Na komunikácia medzi klientom a serverom využíva WSDL	Na odosielanie a prijímanie dát používa XML alebo JSON
3	Volanie servisov prostredníctvom RPC metód	Jednoduché volanie servisov via URL
4	Vracia výsledok, ktorý nie je pre človeka čitateľný	Výsledok je čitateľný, nakoľko sa jedná o čistý XML alebo JSON
5	Tranfer prostredníctvom HTTP, dokáže ale využiť aj iné protokoly, ako napríklad SMTP alebo FTP.	Transfer dát cez protokol, ktorý poskytuje URI.
6	JavaSript dokáže volať SOAP, ale je to zložité na implementáciu	Jednoduché volania prostredníctvom JavaScriptu
7	Pomalšie spracovanie oproti RESTu.	V porovnaní so SOAPom rýchlejšie a jednoduchšie.

Tabuľka 2: Porovnanie SOAPu a RESTu[7]

[7] Zdroj: http://stackoverflow.com/questions/2131965/main-differences-between-soap-and-restful-web-services-in-java [cit. 2017-04-16].

2 RESTOVÉ SLUŽBY

2.1 RESTová architektúra

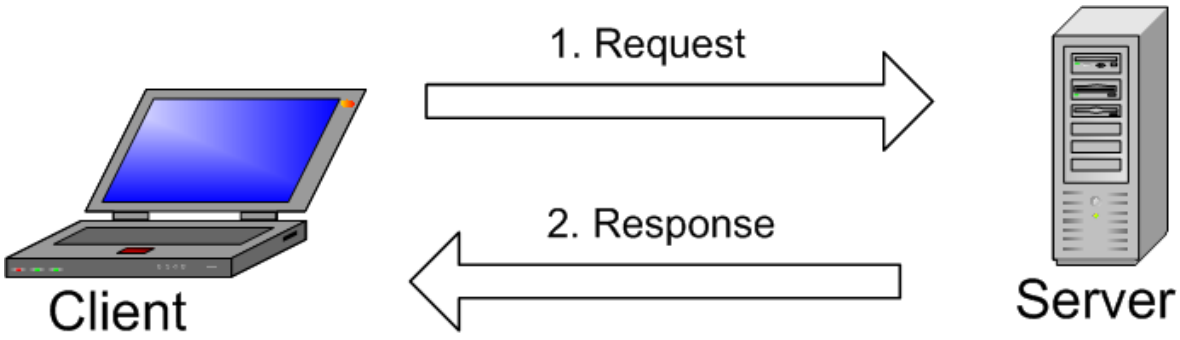
Aby mohla byť architektúra považovaná za RESTovú, musí spĺňať šesť základách bodov, ktoré definoval Fielding vo svojej práci Architectural Styles and the Design of Network-based Software Architectures [5]:

1. Architektúra klient-server
2. Bezstavovosť
3. Cache
4. Jednotné rozhranie
5. Vrstvenie systému
6. Code on Demand

2.1.1 Princípy REST architektúry

2.1.1.1 Architektúra klient-server

Model klient-server je v prípade webových aplikácií jednou z najpoužívanejších architektúr. Princípom je komunikácia medzi klientom a serverom na základne daného rozhrania. Klient pošle na server požiadavku a server požiadavku spracuje a pošle klientovi odpoveď, alebo požiadavku zamietne.



Obrázok 4: Model klient-server[8]

[8] Zdroj: https://i.stack.imgur.com/uKIb7.png [cit. 2017-05-01].

Jednotlivé komponenty tohto modelu by mali byt navrhnuté tak, aby sa ich funkcionality prekrývali čo najmenej. Vďaka tomu môže vývoj na oboch stranách prebiehať nezávisle na sebe za predpokladu, že sa nemení rozhranie medzi nimi.

2.1.1.2 Bezstavovosť

Bezstavovosť komunikácie znamená, že požiadavka musí obsahovať všetky informácie, ktoré sú potrebné na spracovanie tejto požiadavky. To znamená, že v priebehu komunikácie si server nemôže uchovávať žiadny klientský kontext. S tým súvisia ďalšie vlastnosti architektúry, ako sú:

- Škálovateľnosť – tým, že si server nemusí držať informácia o aplikačnom stave, sa zjednodušuje paralelizácia medzi viacerými zariadeniami.
- Viditeľnosť – pre monitorovanie požiadavky stačí vyhodnotiť obsah prichádzajúcej požiadavky.
- Spoľahlivosť – rýchle a jednoduché zotavenie pri čiastočnej chybe.

2.1.1.3 Cache

Cachovanie označuje proces, pri ktorom sú prostredníctvom hardvérovej alebo softvérovej časti počítača uchovávané dáta, za účelom zrýchliť k nim prístup.

Webová cache slúži na ukladanie odpovedí z webových serverov, najmä statického obsahu, čím sa znižuje vyťaženosť siete aj samotných webových serverov.

V prípade RESTu je možné do medzipamäte ukladať predovšetkým zdroje získané pomocou metódy GET, keďže táto metóda slúži pri správnej implementácií výhradne na získavanie zdrojov bez potreby porozumieť obsahu požiadavky. Toto nie je možné napríklad pri technológii SOAP, kde je potrebné požiavku otvoriť, pochopiť jej sémantiku a na základe toho rozhodnúť, ako sa bude daná požiadavka ďalej spracovávať.

Samostné HTTP má v hlavičke uvedené, ako má klient obdržaný zdroj ukladať. Základnou variantou je použitie Expires, kde je uvedený dátum a čas, do kedy je zdroj platný. Podmienkou je, aby klient aj server mali nastavený rovnaký čas. Ďalšou možnosťou je použiť hlavičku Cache-Control, kde je uvedená doba trvania platnosti Cache alebo cachovanie pomocou tzv. ETag. Ten umožňuje v hlavičke odpovedi uviesť informáciu o stave daného zdroja v ľubovoľnom formáte, jedinou podmienkou je, že jeho hodnota musí byť uvedená v uvádzovkách. Keďže sa musí jednať o jednoznačnú identifikáciu

daného zdroja, často sa používajú rôzne hešovacie funkcie. Napríklad v prípade statického obsahu je možné ponechať výpočet ETag hodnoty priamo na aplikačnom servery. Princíp fungovania spočíva v tom, že klient pri prvom dotaze na server obdrží ETag, ktorý následne používa v hlavičke pri nasledujúcich dotazoch. Ak hotnota uvedená v hlavičke požiadavky stále zodpovedná hodnotám servera, vráti server dopoveď 304 Not Modified.

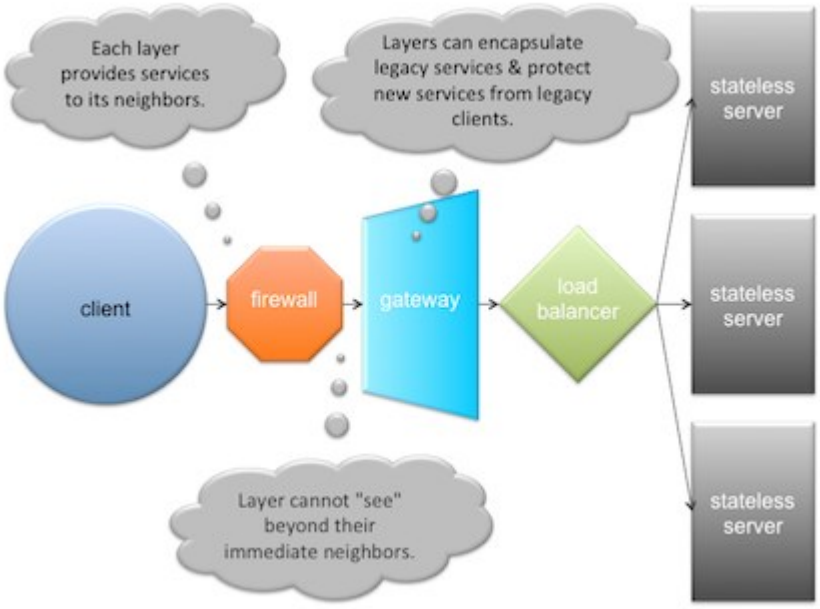
2.1.1.4 Jednotné rozhranie

Aby mohlo byť rozhranie označené ako jednotné, musí splniť nasledujúce podmienky:

- každý zdroj musí mať unikátny identifikátor. K tomuto účelu slúži URI. Medzi zdrojom a URI platí kardinalita 1:n, tzn. že URI identifikuje práve jeden zdroj, ale k jednému zdroju môžu prislúchať viaceré URI. Okrem URI existuje ešte (1) URL, ktorý špecifikuje, akým spôsobom má klient k danému zdroju pristupuje, (2) IRI, ktoré dovoľuje použitie medzinárodných znakových sád a (3) URN, ktorého schému tvorí URI a zvyšok patrí do určitého menného súboru.
- v prípade, že klient získa reprezentáciu zdroja a prislúchajúce metadáta, môže s týmto zdrojom manipulovať,
- správa musí byť dostatočne popisná, aby bolo možné poznať jej sémantický význam,
- a ďalšie stavy aplikácie je možné získať na základe odkazov prijatých v reprezentácií stavu aktuálneho – akronym známy ako HATEOAS (hypermedia as the engine of application state).

2.1.1.5 Vrstvenie systému

Princípom vrstvenia systému je rozdelenie aplikácie na množstvo nezávislých vrstiev s tým, že každá vrstva môže interagovať len s ďalšou najbližšou vrstvou. Problémom pri vrstvení môže byť zvýšená latencia pri spracovaní dát, čo sa aspoň čiastočne dá odstrániť cachovaním.



Obrázok 5: Vrstvenie systému v RESTovej architektúre[9]

2.1.1.6 Code on Demand

Výraz Code on Demand môžeme preložiť ako „kód na vyžiadanie“. REST dovoľuje klientom rozšíriť aplikácia o funkcionalitu, s ktorou sa nepočítalo a to pomocou appletov a skriptov. Veľkou nevýhodou takéhoto rozšírenia je bezpečnostné riziko a zníženie viditeľnosti. Používanie takéhoto kódu je preto dobré poriadne zvážiť a použiť ho len tam, kde výhody vo veľkej miere prevyšujú jeho nevýhody.

2.1.2 Nefunkčné požiadavky RESTového rozhrania[10]

Nefunkčné požiadavky sú požiadavky, ktoré nie sú závislé na konkrétnej implementácii, ale definujú vlastnosti, ktoré by mal mať systém ako celok.

[9] Zdroj: https://apigee.com/about/cdn/farfuture/CSGAvPzBZLywCj_kMv0ixNxuxJ23vFQwpcetpoz7AS8/mtime:1439316218/sites/mktg-new/files/HATEOAS2%20copy.png [cit. 2017-05-03].

[10] Spracované podľa [5].

2.1.2.1 Výkon

Je zásadným parametrom u každého softvéru. V prípade RESTovej architektúry, ktorá je založená na prenose dát po sieti, ovplyvňujú výkon požiadavky samotnej aplikácie, štýl interakcie a nakoniec vzťah medzi jednotlivými komponentmi.

Okrem toho do toho vstupuje aj výkon samotnej siete. Prenos dát ovplyvňuje niekoľko parametrov, ktorými sú priepustnosť, veľkosť dátového toku (tu pripočítavame aj overhead, tzn. dáta nutné k nadviazaniu spojenia. Treba myslieť na to, že časté interakcie nám spôsobujú veľký overhead) a použiteľná šírka pásma.

2.1.2.2 Škálovateľnosť

Škálovateľnosť znamená schopnosť architektúry podporovať veľké množstvo komponentov alebo interakciu medzi komponentmi. Škálovateľnosť sa dá zlepšiť zjednodušením komponentov, distribúciou služieb v rámci mnohých komponentov (decentralizáciou interakcií) a kontrolou interakcií a konfigurácií v dôsledku monitorovania. Škálovateľnosť ovplyvňuje aj frekvencia interakcií, tzn. či je zaťaženie na komponente rovnomerne rozložené v priebehu času, alebo sa vyskytuje v „píkoch“.

Dobre škálovateľná architektúra je taká, ktorá umožňuje priebežné rozširovanie o väčšie množstvo komponentov a nárast interakcií medzi nimi.

2.1.2.3 Jednoduchosť

Základným spôsobom, ktorým je možné docieliť jednoduchosť pomocou architektonického štýlu, je uplatňovanie princípu oddelenia zodpovedností v rámci komponentov. Ak môžu byť funkcie implementované tak, že jednotlivé komponenty sú menej zložité, potom budú ľahšie pochopiteľné a implementované. Rovnako tak takéto oddelenie uľahčuje úlohu uvažovať o celkovej architektúre.

2.1.2.4 Modifikovateľnosť

Požiadavky na aplikáciu sa v čase môžu meniť. Dobre modifikovateľný systém by mal tieto zmeny zvažovať a umožňovať.

2.1.2.5 Viditeľnosť

Viditeľnosť v tomto prípade odkazuje na schopnosť komponentu monitorovať alebo sprostredkovať interakciu medzi dvoma ďalšími komponentmi. Viditeľnosť umožňuje zvýšiť

výkon prostredníctvom zdieľaného ukladania do pamäte interakcií, škálovateľnosti prostredníctvom vrstvových služieb, spoľahlivosti prostredníctvom reflexného monitorovania a zabezpečenia tým, že umožňujú, aby boli interakcie kontrolované mediátormi (napríklad sieťovými firewallmi).

2.1.2.6 Prenositeľnosť

Aplikáciu považujeme za prenositeľnú, pokiaľ je schopná fungovať aj v inom prostredí, než v akom bola vyvinutá. Iným prostredím rozumieme napr. iný operačný systém.

2.1.2.7 Spoľahlivosť

Spoľahlivosť z pohľadu aplikačných architektúr možno považovať za mieru, do akej je architektúra náchylná k zlyhaniu na systémovej úrovni za prítomnosti čiastkových porúch v rámci komponentov, konektorov alebo dát. Dobre navrhnutá architektúra môže zlepšiť spoľahlivosť tým, že sa vyhýba slabým miestam, umožňuje redundanciu a monitorovanie alebo znižuje rozsah neschopnosti obnoviť činnosť.

2.1.3 Dátové jazyky

S dátovými jazykmi súvisia pojmy zdroj (alebo resourse) a reprezentácia tohto zdroja.

Zdroj je základný prvok architektonického štýlu REST a rozumieme pod ním ľubovoľný logický objekt, s ktorým môžeme manipulovať pomocou HTTP metód.

Reprezentácia zdroja znamená tranformáciu tohto zdroja do podoby, ktorú môžeme definovať ľubovolným dátovým formátom, pomocou ktorého môžeme prenášať informácie. Každý zdroj je možné identifikovať pomocou URI.

Väčšina RESTových rozhraní používa JSON aleb XML reprezentáciu. Pre obidve dátové formáty existujú schémy, ktoré opisujú ich štruktúru, a tým zabezpečujú bezproblemové používanie. Pre obe reprezentácie tiež platí, že sa jedná o otvorený formát, tzn. že nie je zviazaný so žiadnou platformou alebo proprietárnou technológiou.

2.1.3.1 XML

XML je rovnako ako JSON jazyk, ktorý sa radí do skupiny značkovacích a samopopisných jazykov, tzn. že okrem samotného obsahu dokumentu popisuje aj jeho štruktúru. Túto štruktúru popisuje z hľadiska vecného obsahu, nerieši úpravu vzhľadu.

Súčasťou XML dokumentu sú nasledujúce prvky:

- Hlavička dokumentu – alebo aj XML deklarácia, je prvým riadkom XML dokumentu. Skladá sa z kľúčového slova xml a povinnej deklarácie verzie.
- Tagy – je značka, pomocou ktorej štrukturujeme XML dokument. Sú uzatvorené do ostrých zátvoriek, rozlišujeme začiatočné a koncové tagy (napr. <meno></meno>).
- Elementy – element je základný prvok XML. Aby bol dokument považovaný za štruktúrovaný, musí mať práve jeden koreňový element, neprázdne elementy musia byť ohraničené začiatočným a koncovým tagom, elementy môžu byť vnorené, alebo nemôžu sa prekrývať.
- Atribúty – jedná sa o doplňujúcu informáciu k elementom. Zapisujú sa do začiatočného tagu v tvare meno atribútu, rovná sa a hodnota atribútu zapísaná v úvodzovkách.
- Znakové a textové entity – používajú sa miesto znakov, ktoré nemôžeme zapísať do textu.
- CDATA – *character data* sa používajú na zápis veľkých častí textu so špeciálnymi znakmi.
- Komentáre – nie sú spracované programom.
- Procesné inštrukcie – umožňujú pridávať neštandardné dáta k XML dokumentu. Pomocou nich môžeme do kódu vložiť napr. css súbory, alebo príkazy skriptovacieho jazyka.

Spracovanie XML najčastejšie prebieha dvoma spôsobmi a to:

- DOM parser vyrobí z XML obraz v pamäti;
- SAX parser prechádza XML postupne.

2.1.3.2 JSON

JSON je odľahčený formát určený pre výmenu dát. Navrhol ho Douglas Crockford a jeho presnú špecifikáciu je možné nájsť v RFC 4627[11]. Odľahčený znamená, že na rozdiel od XML nie je v jeho prípade potrebné používať DOM, prítomnosť uzlov obsahujúcich len biele znaky a pod.

Používa sa na zápis krátkych štruktúrovaných dát.

---

[11] Dostupné na adrese https://tools.ietf.org/html/rfc4627.

Pomocou JSON môžeme prenášať nasledujúci typ dát:

- JSONString – textový reťazec;
- JSONNumber – číslo;
- JSONBoolean – logická hodnota;
- JSONNull – hodnota null;
- JSONArray – pole;
- JSONObject – objekt.

2.1.3.3 Porovnanie XML a JSON zápisu

JSON zápis	**XML zápis**

{"menu": {
"id": "file",
"value": "File",
"popup": {
"menuitem": [
{"value": "New", "onclick":
"CreateNewDoc()"},
{"value": "Open", "onclick":
"OpenDoc()"},
{"value": "Close", "onclick":
"CloseDoc()"}
]
}
}}
	<menu id="file" value="File">
	<popup>
	<menuitem value="New"
	onclick="CreateNewDoc()" />
	<menuitem value="Open"
	onclick="OpenDoc()" />
	<menuitem value="Close"
	onclick="CloseDoc()" />
	</popup>
	</menu>

Tabuľka 3: Porovnanie XML a JSON zápisu

2.1.4 CRUD

CRUD je skratka zložená zo začiatočných písmen slov create, read, update a delete. Jedná sa o štyri základné operácie, pomocou ktorých budeme vykonávať zmeny nad perzistným úložiskom, tzn. databázou. Každé slovo môže byť namapované na sql výraz, http metódu alebo dds operáciu.

Operácia	**SQL**	**HTTP**	**DDS**
Create	INSERT	PUT / POST	write
Read	SELECT	GET	read / take
Update	UPDATE	PUT / POST / PATCH	write

Delete	DELETE	DELETE	dispose

Tabuľka 4: Namapovanie CRUD operácií na sql, http a dds

2.1.5 Koncové body (endpoint)

Koncové body sú URI, na ktorých sú prostredníctvom API dostupné zdroje. Na ilustráciu použijem príklad s fiktívnou prevádzkou Restbuck z knihy REST in Practice [4]. Pod URI http://restbuck.com/order budú dostupné všetky objednávky, konkrétna objednávka bude potom dostupná pod koncovým bodom /order/{order_id}. Zápis v zložených zátvorkách predstavuje dosaditeľný parameter, v našom prípade pôjde o konkrétnu objednávku kávy. URI nám dovoľuje špecifikovať ďalšie parametre, ktoré nám slúžia napr. pre prístup ku kolekcii veľkého množstva údajov. Napríklad prostredníctvom http://restbucks.com/order/{year}/{month}/{day} sa dostaneme ku všetkým objednávkam pre daný rok, mesiac a deň.

Vzťah medzi koncovými bodmi a HTTP metódami uvádzam v Tabuľke 1: Metódy HTTP pre webové služby, ktoré sú RESTful.

2.2 Komunikácia prostredníctvom RESTových služieb

Napriek tomu, že HTTP nie je jediný komunikačný protokol a RESTová architektúra nie je pevne definovaná nad ním, budem ďalej pracovať výhradne s týmto protokolom. Dôvodom je, že sa jedná o hlavný komunikačný protokol internetu.

2.2.1 HTTP medódy

HTTP metóda definuje, čo sa má nad daným zdrojom vykonať.

GET – žiadosť o zaslanie dát s tým, že sama môže zasielať doplňujúce dáta, ako napríklad session id a pod.

HEAD – ako GET, ale nezasiela doplňujúce dáta.

POST – odosiela užívateľské dáta na server, napr. pri vyplnení webového formulára.

PUT – umožňuje nahrať dáta na server.

DELETE - umožňuje zmazať dáta zo servera.

TRACE – klientovi pošle späť prijatý request, klient teda môže vidieť, ako bola jeho pôvodná požiadavka menená pri prechode cez jednotlivé servre pri prechode k cieľovému serveru.

OPTIONS – umožní overiť podporované HTTP metódy koncového zdroja.

CONNECT – spojenie cez konktrétny definovaný port. Používa sa pri prechode cez proxy.

Uvedené metódy môžeme rozdeliť na bezpečné a idempotentné.

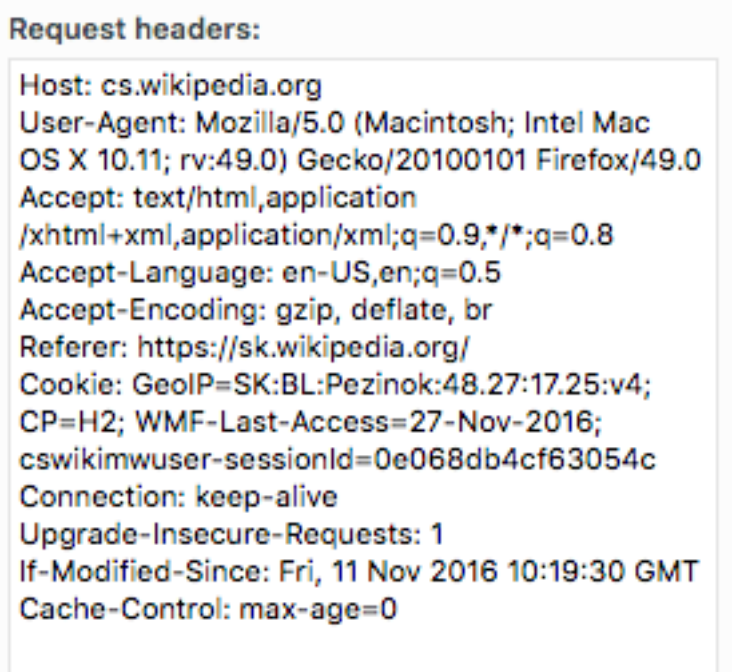
Medzi bezpečné medódy patria GET, HEAD, OPTIONS a TRACE, pretože slúžia na získavanie informácií a nemali by meniť stav servera. Naopak, metódy POST, PUT a DELETE môžu stav servera meniť (odoslať email, vykonať finančnú transakciu a pod.), preto by nemali byť používané webovými vyhľadávačmi, ako napr. Google. Toto rozdelenie je však iba orientačné, pretože pri nesprávnej implementácii môžeme aj požiadavkou GET vložiť napr. riadok do databázy.

To, že je metóda idempotentná znamená, že aj niekoľkonásobné prevedenie rovnakej požiadavky bude mať rovnaký výsledok, ako prevedenie jednej požiadavky. Do tejto skupiny metód patrí PUT a DELETE a všetky bezpečné metódy, ktoré boli uvedené vyššie.

2.2.2 HTTP hlavičky

Hlavičky sú dôležitou súčasťou HTTP requestov, pretože definujú požiadavky na server, ako napríklad preferovaný jazyk, kódovanie a pod.

Príklad pripojenia na https://cs.wikipedia.org/wiki/Webová_služba:



Obrázok 6: Príklad HTTP hlavičky

2.2.3 HTTP statusové kódy

Odpoveď servera začína špecifikáciou verzie protokolu HTTP x.x. V ďalšom riadku je odpoveď, či bola žiadosť spracovaná bez problémov, alebo došlo k nejakej chybe, ďalej nasleduje hlavička, prázdy riadok a samotné dáta.

Stavové informácie môžu nadobúdať nasledovné hodnoty:

- 1xx – informačný stavový kód;
- 2xx – požiadavka od klienta bola úspešne spracovaná;
- 3xx – presmerovanie. Požiadavka si vyžaduje dodatočnú interakciu klienta;
- 4xx – chyba na strane klienta;
- 5xx – chyba na strane servera.

2.3 Bezpečnosť RESTových služieb

RESTové služby sú vytvorené tak, aby boli verejne dostupné a jednoduché na používanie. To ich však predurčuje k veľkej zraniteľnosti. Cieľom útočníka nemusí byť iba získanie prístupu k takejto službe, ale aj :

- Narušenie služby;
- Odpočúvanie služby;
- Predstieranie falošnej identity;
- Získanie riadenia platformy.

Za účelom chrániť dáta, ktoré sú prenášané v rámci webových služieb, vznikli rôzne mechanizmy riadenia prístupu.

V rámci bezpečnosti webových služieb ako takých, sa používajú pojmy entita a identita. Pod entitnou si môžeme predstaviť človeka, aplikáciu, webovú službu a pod. Identita je potom entita v určitej bezpečnostnej doméne, v rámci ktorej sú stanovené bezpečnostné pravidlá, napr. k akým informáciám má identita prístup.

Ak bola identita vytvorená a overená v jednej bezpečnostej doméne a chceme, aby bola uznaná v rámci druhej bezpečnostnej domény s rovnakými právami, hovoríme o portovaní, resp. preklade takejto identity. Jedným z riešení je vytvorenie vzťahu dôvery medzi týmito doménami. Dôvodom, prečo je potrebne riešiť tento problém je, že v prípade RESTových služieb dochádza k prenosu informácií medzi bezpečnostnými doménami a je dôležité zaistiť utajenie a integritu prenášaných informácií.

Na zabezpečenie existuje päť základných bezpečnostných systémov a to:

1. Šifrovanie – transformácia informácie do podoby, ktorú môžu prečítať iba entity, pre ktoré je táto informácia určená. Šifrovanie sa uskutočňuje prostredníctvom šifrovacích algoritmov. Existujú dva typy šifrovania:
   - Symetrické – na šifrovanie a dešifrovanie použijeme ten istý kľúč;
   - Asymetrické – využíva sa súkromný a verejný kľúč.
2. Digitálny podpis – súčasťou posielanej správy je aj heš. Prijímajúca strana si následne vypočíta heš prijatej správy. Zhoda prijatého a vypočítaného hešu by mala zaručovať integritu prijatej správy.
3. Autentifikácia – slúži na identifikovanie a overenie entity.
4. Autorizácia – slúži na overenie, ktorá identita má právo pristupovať ku ktorej informácii, a čo môže s danou informáciou robiť (napr. only read, read – write).
5. Audit – uchovávanie záznamov o prístupoch k informáciám.

Implementácia bezpečenostných opatrení je v prípade RESTových služieb potrebná ako na strane prevádzkovateľa webových serverov, tak na strane vývojára. O uvedených bezpečnostných opatreniach, ktoré je potrebné zabezpečiť v rámci nárhu RESTovej služby, si podrobnejšie povieme v tejto podkapitole.

2.3.1 Bezpečnostné hrozby pri implementácii webových služieb

Bezpečnostné opatrenia pri implementácii webových služieb musia byť vždy navrhnuté v súlade so znalosťami na zabezpečovaný systém. Nie všetky opatrenia, štandardy a dostupné techniky sú v rámci tej ktorej inštitúcie vyhovujúce. Napríklad banka bude mať iné požiadavky na svoje systémy z hľadiska bezpečnosti, ako e-shop s topánkami. Je potrebné vždy zvážiť, aké reálne hrozby webovýh službám hrozia a na základe toho implementovať konkrétne bezpečnostné opatrenia.

Medzi najrozšírenejšie hrozby patria:

- Modifikácia správy – útočník vloží, odstráni alebo modifikuje časť správy;
- Strata dôveryhodnosti – informácia je zverejnená neautorizovanej entite;
- Falošné správy – vytvorenie a odoslanie falošnej správy tak, aby ju príjemca považoval za správu od platného odosielateľa;
- Man-in-the-middle - komunikácia prechádza cez tretiu osobu bez toho, aby o tom odosielateľ a prijímateľ správy vedeli;

- Falšovanie odosielateľa – odosielateľ vytvorí a odošle správu s údajmi, ktoré patria inému autorizovanému odosielateľovi;
- Falošná požiadavka - odosielateľ vytvorí správu s falošnými autentizačnými údajmi;
- Opakovanie správy – útočník opakovane odosiela už predtým poslanú správu;
- Opakovanie časti správy - útočník vloží jednu alebo viacero častí z predchádzajúcej správy do novej správy;
- Odmietnutie služby – útočník spôsobí, že systém neobslúži zaslanú platnú požiadavku.

	Modifikovanie správy	**Strata dôverihodnosti**	**Falošné správy**	**Man-in-the-Middle**	**Falšovanie odosielateľa**	**Falošná požiadavka**	**Opakovanie správy**	**Opakovanie časti správy**	**Odmietnutie služby**
Šifrovanie XML		x		x	x	x		x	
Podpisovanie XML	x		x		x	x	x	x	
Tokeny WS-Security			x		x	x			
WS-Addressing IDs							x		
SSL/TLS	x	x	x	x	x	x		x	
SSL/TLS s autentizáciou klienta	x	x	x	x	x	x		x	
HTTP Autentizácia			x		x	x			

Tabuľka 5: Zoznam aktuálnych štandardov a hrozby, ktoré pokrývajú[12]

[12]Zdroj: https://www.csirt.gov.sk/doc/MFSRVzdelavanie/01Vzdelavanie2014/10Prezentacie/Bezpecnost_webovych_sluzieb.pdf [cit. 2017-04-16].

2.3.2 Príklady technológií na podporu bezpečnosti

2.3.2.1 Autentifikácia

Slúži na identifikovanie a overenie entity. Poznáme rôzne druhy autentifikácie.

BASIC autentifikácia

Basic autentifickácia predstavuje jednoduché overenie prístupu na webové stránky. Postup je nasledovný. Webový server vyzve pomocou HTTP protokolu klienta, aby poslal aj svoje autentizačné informácie, tzn. meno a heslo.

Výhodou tejto metódy je ľahká implementácia. Meno a heslo sa ale prenáša nešifrované, takže je ľahkým terčom pre MiM útok. Server tiež nemá možnosť klietovi prikázať, aby sa odhlásil.

Príklad komunikácie:

1. Klient požiada o stránku, ktorá vyžaduje basic autentifikáciu
2. Server odpovie 401 (authorization required) a pokytne informáciu o požadovanom spôsobe autentifikácie
3. Klient sa spýta žiadateľa na meno a heslo. Užívateľ meno a heslo zadá a opäť pošle požiadavku na server, doplnenú o autentizačné údaje.
4. Ak bolo meno a heslo uvedené správne, server pošle požadované informácie klientovi.

DIGEST autentifikácia

Digest autentifikácia je podobná basic autentifikácii s tým rozdielom, že prihlasovacie údaje nie sú prenášané v kódovaní Basic, ale sú hešované. Tento spôsob nie je veľmi rozšírený.

Autentifikácia prostredníctvom certifikátov, verejných kľúčov a PKI

Autentifikácia prostredníctvom certifikátov, verejných kľúčov a PKI slúži na vytvorenie bezpečného a dôverného komunikačného kanála medzi aplikáciou a webovým serverom.

PKI – Public Key Infrastructure – jedná sa o infraštruktúru, ktorá zabezpečuje zverejňovanie verejných kľúčov a ich správu. Základné role, ktoré vystupujú v PKI sú:

- Držiteľ certifikátu – entita, ktorej identita je certifikovaná;
- Certifikačná autorita – vydavateľ certifikátov;

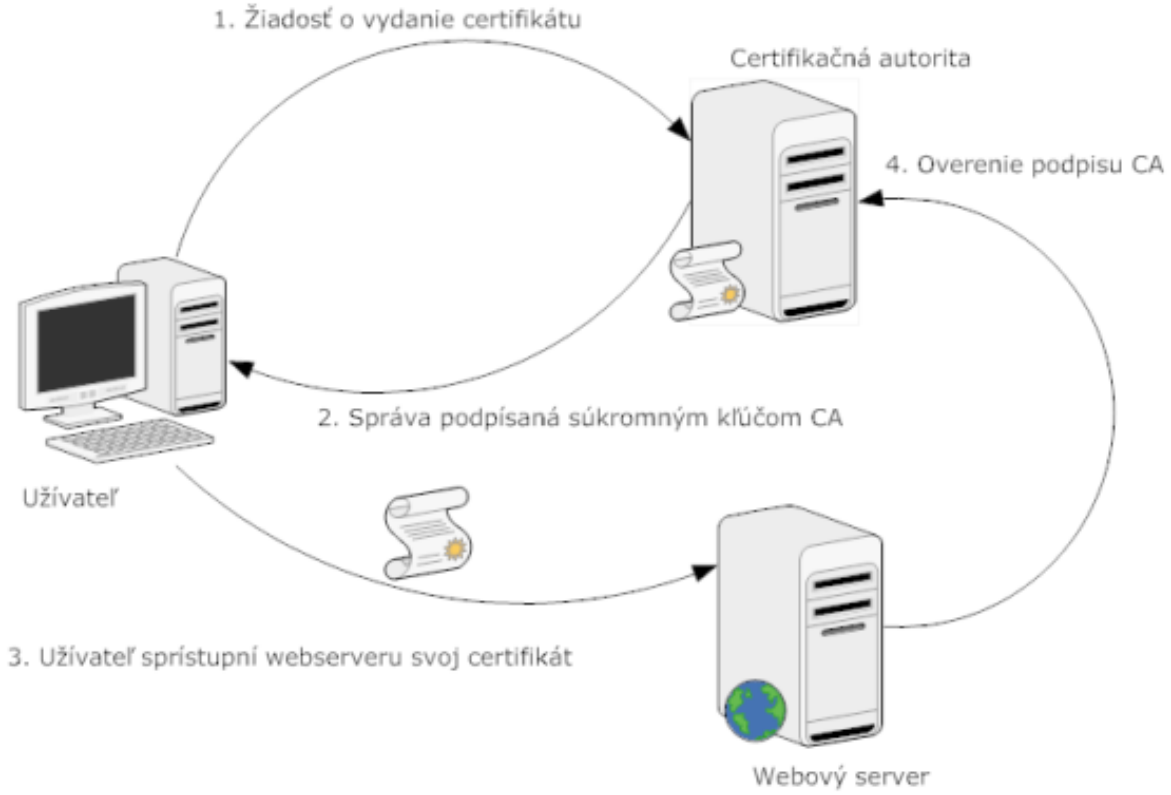
- Užívateľ certifikátu – strana, ktorá sa spolieha na dôveryhodnosť certifikátu.

Poznáme rôzne typy digitálnych certifikátov. Líšia sa nielen sémantikou, ale aj syntaxou. Medzi základné typy patria digitálny certifikát osoby, ktorý identifikuje konkrétnu osobu a prideľuje jej určité privilégia. Takýto certifikát umožňujú používať všetci významní poštoví klienti. Druhým typom je digitálny certifikát servera, ktorý potvrdzuje identitu servera.

PKI	Charakteristika CA	Typ identifikácie
X.509	Hierarchia autorít, zodpovedných za identifikáciu, vzájomná certifikácia	Definitoricky globálna (Distinguished Name, X.500), prakticky lokálna, volená vydávajúcou CA so snahou o unikátnosť
PGP	Pavučina dôvery – násobné cesty certifikácie s cieľom zaistiť prijateľnú dôveryhodnosť amatérskych vydavateľov certifikátov	Globálna – globálne unikátna e-mail adresa (vďaka „Domain Name System“), nemusí byť však perzistentná
SPKI/SDSI	Jedna identifikačná autorita, nie je nutná CPS	Lokálna
SPKI bez mien	Hierarchia autorizačných autorít, volitelný princíp „k-of-n“	Globálna – verejnú kľúč alebo jeho globálne unikátny a perzistetný hash

Tabuľka 6: Základné druhy PKI[13]

[13] Zdroj: http://www.fi.muni.cz/usr/staudek/vyuka/security/stud_lit/D01_C.pdf [cit. 2017-04-17].



Obrázok 7: Proces vydania certifikátu[14]

HTTP komunikácia sa najčastejšie šifruje prostredníctvom SSL resp. TLS protokolu. Pre vydanie certifikátu pri obojstrannej autorizácii sa využíva PKI X.509, spomínaný v Tabuľke 3: Základné druhy PKI.

2.3.2.2 Autorizácia

Pomocou autorizácie overujeme, či má žiadateľ služby oprávnenie vykonať operáciu, o ktorú sa pokúša, a k akým zdrojom má prístup, či už sa jedná o databázy alebo iné typy súborov.

Jedným zo spôsobov autorizácie je využitie konfiguračného súboru web.config webovej služby, kde definujeme pravidlá, ktoré sú vykonávané prostredníctvom HTTP modulu UrlAuthorization-Module. Princíp fungovania je založený na analýze a kontrole každej požiadavky užívateľa.

[14] Zdroj: http://www.kiwiki.info/index.php/S%C3%BAbor:Dp_2010_jm_5.png [cit. 2017-04-17].

2.3.2.3 XML Security

XML Security znamená rozšírenie XML o bezpečnostné mechanizmy. Tvorí ho päť častí:

1. XML Signature – digitálny podpis aplikovaný na XML dáta. Zabezpečuje autentifikáciu, kontrolu integrity a dôvernosti dát a slúži na podporu neopakovateľnosti.
2. XML Encryption – štandard, ktorý opisuje spôsob vloženia výsledku šifrovania XML dát.
3. Security Assertion Makrup Language – štandard, ktorú umožňuje zdieľanie autentifikačných a autorizačných informácii v rámci systému, ako sú napríklad role a certifikáty.
4. XML Access Control Markup Language – úzko súvisí so Security Assertion Makrup Language. Slúži ako mechanizmus na šírenie autentifikačných a autorizačných informácii

XML Key Managment Specification – protokol, ktorý špecifikuje hospodárenie s verejnými kryptografickými kľúčmi.

3 ANALÝZA FRAMEWORKOV PRE TVORBU RESTOVÝCH ROZHRANÍ

REST architektúra sa neviaže na žiadny programovací jazyk. Na jej implementáciu môžeme použiť ktorýkoľvek z dostupných frameworkov, napríklad zo sveta Javy, Node.js, Pythonu, PHP alebo Ruby. V tejto kapitole uvedieme stručnú analýzu vybraných frameworkov. Zoznam bol zostavený na základe obľúbenosti na diskusných fórach, ako je stackoverflow.com alebo quora.com a tiež podľa prieskumu, ktorý realizoval optimalbi na svojom blogu v rámci série What's the best restful web api framework[15].

3.1 Spring (Java)

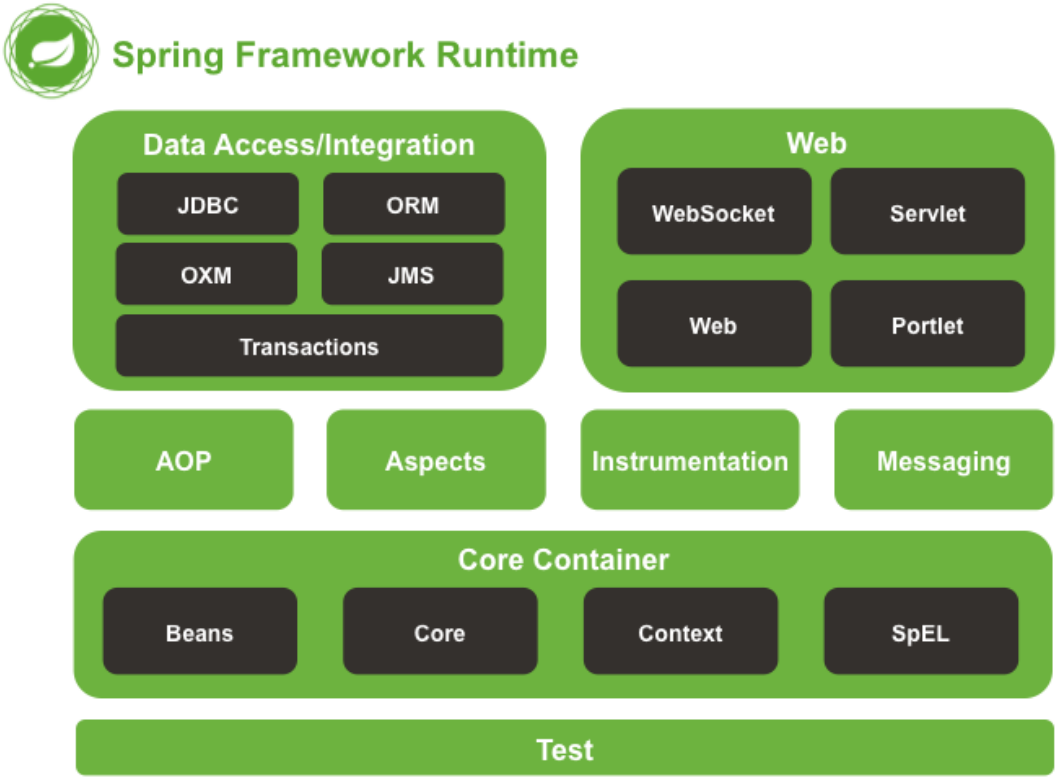
Spring je open-source framework pre vývoj J2EE aplikácií a môže byť použitý v rámci ľubovolnej Java aplikácie.

Skladá sa z častí, ktoré su organizované do modulov. Moduly sú rozdelené do nasledujúcich skupín:

- Core Container – obsahuje moduly, ktoré poskytujú základné časti frameworku, prostriedky pre prístup k objektom a rozšírenie jednotného jazyka pre vytváranie výrazov.
- Data Access/Integration – moduly v rámci tejto skupiny poskytujú integračné vrstvy pre relačné mapovanie, funkcie pre tvorbu a príjem správ a pod.
- Web – obsahuje moduly pre základné webovo orentované funkcie a springovú implementáciu MVC.
- Aspect Oriented Programming – je modul, ktorý implementuje podporu pre aspektovo orientované programovanie. Umožňuje separovať časti kódu, ktoré sa vinú celou aplikáciou (ako napríklad autorizácia alebo logovanie) do tzv. aspektov a ich aplikáciu do POJO objektov.
- Instrumentation

[15] Prvá časť je dostupná na http://optimalbi.com/blog/2016/07/07/whats-the-best-restful-web-api-framework-part-1/ [cit. 2017-05-05]. Na konci príspevku sa nachádzajú odkazy na ďalšie časti.

- Test – tento modul podporuje testovanie jednotlivých komponentov prostredníctvom JUnit alebo TestNG.



Obrázok 8: Architektúra Spring frameworku[16]

3.2 Express (Node.js)[17]

Express.js, alebo jednoducho Express, je open-source framework, určený na vytváranie webových aplikácií a API, postavený na Node.js. V podstate sa jedná o server framework pre Node.js.

Pôvodný autor TJ Holowaychuk ho opísal ako server inšpirovaný Sinatrou[18], čo znamená, že je relatívne minimálny, a zložitejšie funkcie sú k dispozícii ako pluginy.

---

[16] Zdroj: https://acntech.no/content/images/2016/10/spring-overview.png [cit. 2017-05-05].
[17] Dokumentácia je dostupná na https://expressjs.com/
[18] Sinatra je open-source webový framework a DSL napísaný v Ruby. Jeho hlavnou prioritou je malá veľkosť, jednoduchosť a flexibilita. Nepoužíva MVC, ale rovno mapuje kód na URL.

Stručná charakterisktika Expressu:

- Je to cross-platformový framework,
- Je to Node.js server-side framework, určený na vývoj webových a mobilných aplikácií,
- Je napísaný v jazyku JavaScript,
- Je navrhnutý na vytváranie jednostránkových, viacstránkových a hybridných mobilných a webových aplikácií,
- Je to efektívny nástroj pre API založené na JSONe,
- Umožňuje vytvárať real-time webové aplikácie.

Začiatočné písmeno slova Express je aj súčasťou skratky MEAN. MEAN Stack je open-source JavaScriptový stack, určený na vytváranie dynamických webových stránok a webových aplikácií.

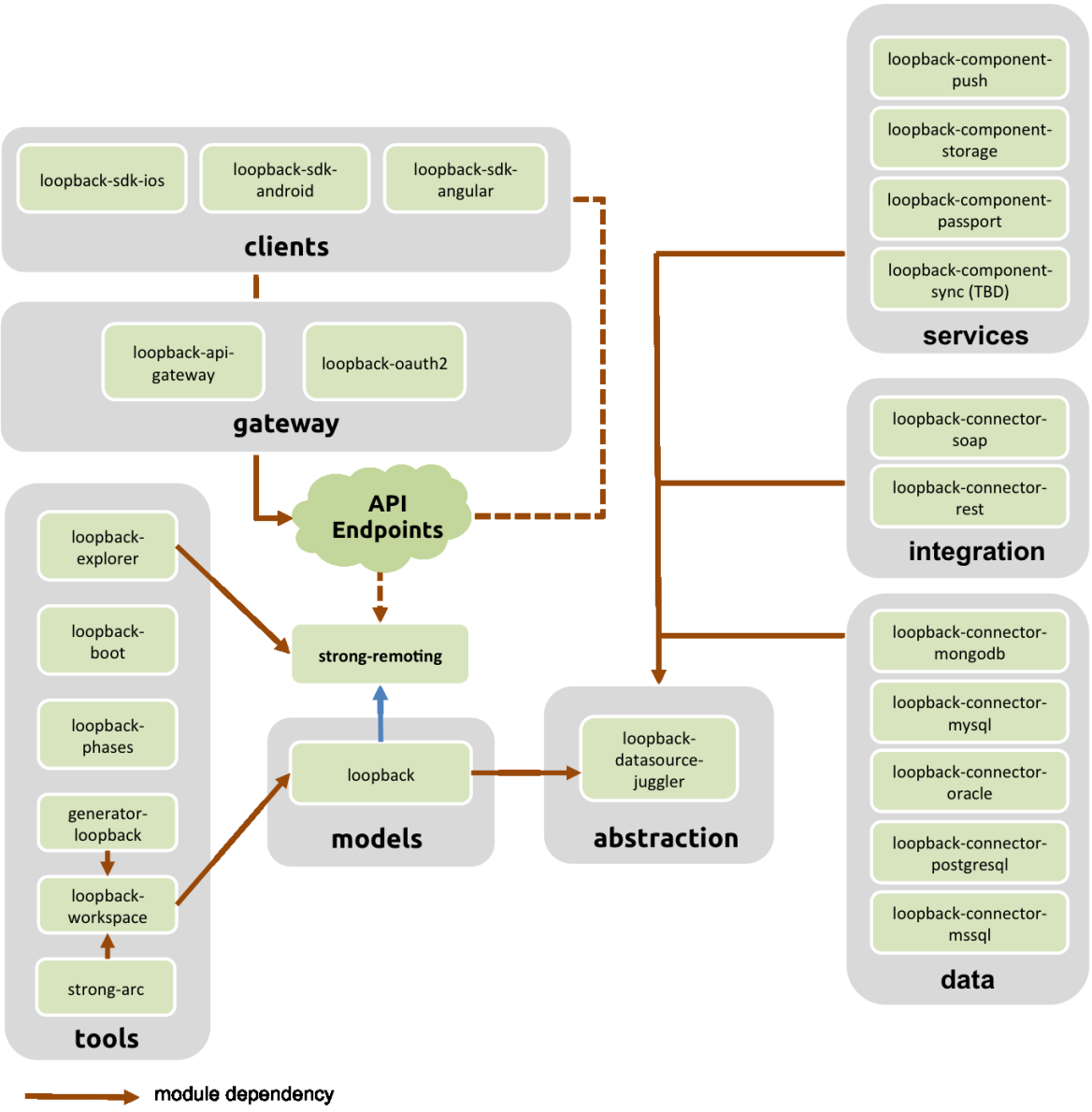
Komponenty MEAN stacku sú nasledujúce:

- MongoDB – noSQL databáza,
- Express.js – framework pre vytváranie webových aplikácia, ktorý beží na Node.js,
- Angular – JavaScript MVC framawork určený na vývoj FE,
- Node.js – určený predovšetkým na vývoj webových serverov.

3.3 Loopback (Node.js)[19]

Loopback je open-source framework, ktorý obsahuje súbor Node.js modulov pre rýchle a jednoduché vytvorenie aplikácií, ktoré používajú RESTovú architektúru. V základnej inštalácii sú už preddefinované niektoré moduly, ako napríklad User, Role, ACL a RoleMapping. Ďalšie je možné definovať jednoducho podľa špecifikácie.

[19] Dokumentácia je dostupná na https://loopback.io/



Obrázok 9: Vzťah medzi kľúčovými LoopBack modulmi[20]

3.4 Django (Python) [21] [22]

Django je open-source webový framework, napísaný v jazyku Python. Ponúka možnosť automatickej tvorby administrácie projektu, ktorá sa generuje dynamicky podľa dátového modelu.

[20] Zdroj: http://loopback.io/images/9830413.png [cit. 2017-05-05].
[21] Dokumentácia je dostupná na https://docs.djangoproject.com/en/1.11/
[22] Spracované podľa https://cs.wikipedia.org/wiki/Django

Framework poskytuje:

- Objektovo-relačný mapper,
- Systém zobrazenia pre spracovanie požiadaviek,
- Jednoduchý webový server ako pre vývoj, tak pre testovanie,
- Validačný systém pre formuláre,
- Rôzne metódy cachovania,
- Komunikačný systém medzi komponentami,
- Serializačný systém pre XML a JSON.

Požiadavka je v Djangou reprezentovaná ako inštancia triedy HttpRequest. Atribúty triedy HttpRequest sú nasledovné:

- method - textový reťazec, ktorý reprezentuje HTTP metódu požiadavky.
- path - cesta k požadovanej stránke.
- GET - atribúty poslané v URL.
- POST - atribúty poslané v tele požiadavky.
- REQUEST - všeobecný prístup k atribútom.
- COOKIES - cookies posielané HTTP požiadavkou.
- FILES - Súbory na uploadovanie.
- META - obsahuje všetky dostupné HTTP hlavičky.
- user - objekt overeného používateľa.

Odpoveď reprezentuje inštancie triedy HttpResponse.

Medzi prijatím požiadavky a odoslaním odpovede z konkrétneho view prejde požiadavka cez reťaz tzv. Middleware - jedná sa o aplikáciu návrhového vzoru Chain of Responsibility[23].

3.5 Flask (Python) [24]

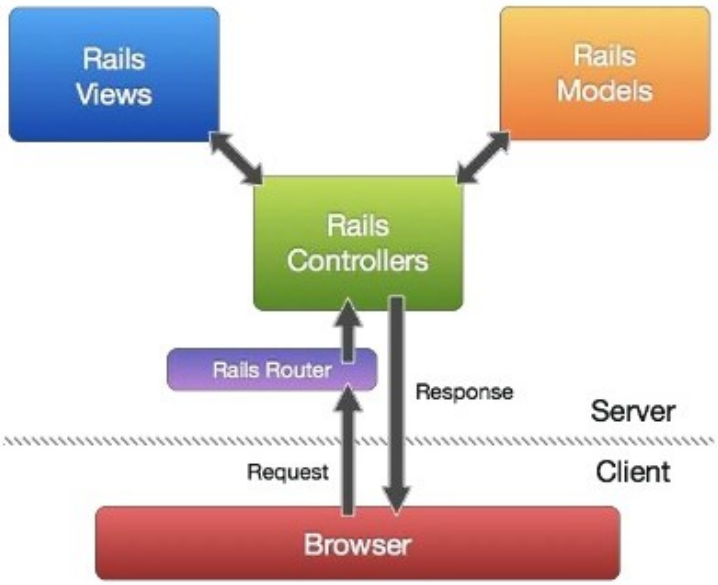
Flask je micro webový framework. Micro znamená, že Flask štandardne neobsahuje abstraktnú databázovú vrstvu, validáciu formulárov alebo čokoľvek iné, čo je už dostupné

[23] Príklad implementácie návrhového vzoru Chain of Responsibility je možné nájsť napríklad tu https://www.tutorialspoint.com/design_pattern/chain_of_responsibility_pattern.htm
[24] Dokumentácia je dostupná na http://flask.pocoo.org/docs/0.12/

v rôznych knižniciach. Namiesto toho Flask podporuje rozšírenia, ktoré umožňujú pridať takúto funkčnosť do aplikácie, ako keby bola implementovaná v samotnom Flasku. Početné rozšírenia poskytujú integráciu databázy, overovanie formulára, upload dokumentov, rôzne autentifikačné technológie a ďalšie.

3.6 Rails (Ruby)

Ruby on Rails (alebo jednoducho iba Rails) je framework na vývoj webových aplikácií, ktoré využívajú pripojenie na databázu. Používa architektúru MVC.



Obrázok 10: Rails – štruktúra aplikácie[25]

Rails je založený na jazyku Ruby, tzn. Ajax v šablonách pre zobrazenie, odpovede v riadiacej aplikácii a architektúre v modeloch, ktoré obaľujú databázu. K spúšťaniu aplikácie je potrebná iba databáza.

Jeden zo základných princípov Rails je, že programátor konfiguruje iba tie časti aplikácie, ktoré sa odlišujú od bežných nastavení.

[25] Zdroj: https://image.slidesharecdn.com/rubyror-140609115622-phpapp01/95/ruby-on-rails-penetration-testing-5-638.jpg?cb=1402316148 [cit. 2017-05-05].

II. PRAKTICKÁ ČÁST

4 ANALÝZA POŽIADAVIEK

Cieľom analýzy požiadaviek je zistiť, aký problém treba riešiť a aké sú požiadavky na aplikáciu. Na základe týchto vstupných údajov sa potom rozhoduje nad vhodnými technológiami.

V mojom prípade bolo požiadavkou vytvoriť RESTové API pre prístup do databázy a následne jednoduchú aplikáciu, na ktorej bude možné prezentovať funkčnosť tohto API.

Na základe funkčných a nefunkčných požiadaviek na systém boli následne stanovené použité technológie, ktorým sa venujem v nasledujúcej kapitole.

4.1 Funkčné požiadavky

Na funkčné požiadavky je možné pozerať sa, ako na súbor požiadaviek konečného užívateľa a na druhej strane, ako na súbor požiadaviek na API rozhranie.

Konečnému užívateľovi systém umožní:

- prezerať si zoznam úloh v dvoch pohľadoch a to z pohľadu plánovaných úloh a z pohľadu ukončených úloh,
- editovať plánované úlohy,
- mazať plánované aj ukončené úlohy,
- meniť stav úloh, tzn. plánovaná do stavu ukončená, ale aj opačne.

Súbor funkčných požiadaviek na API rozhranie:

- realizovať CRUD operácie nad dátovou schémou.

4.2 Nefunkčné požiadavky

Medzi nefunkčné požiadavky patrí:

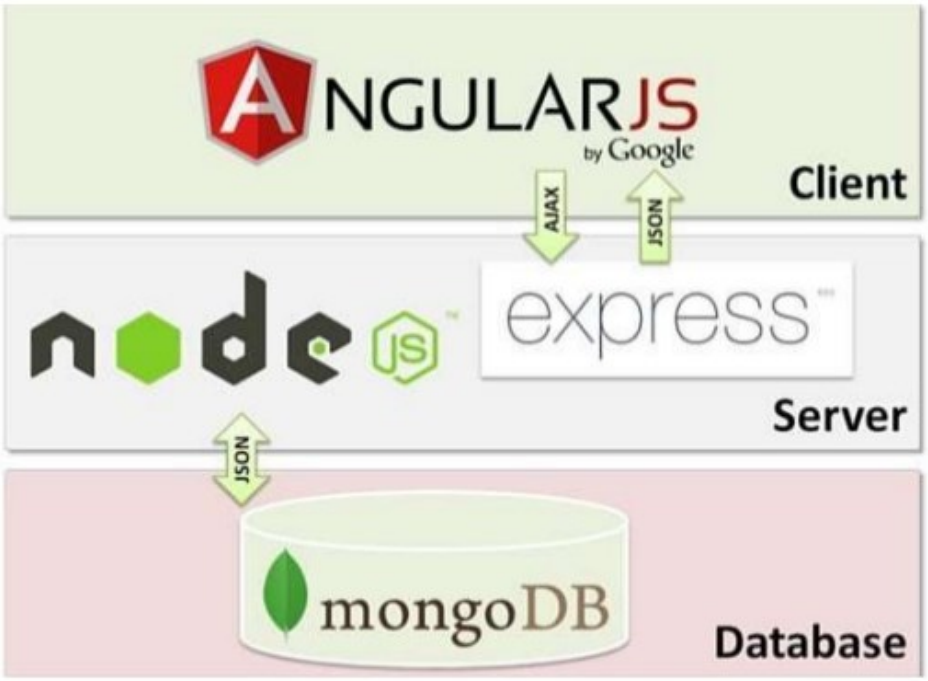
- RESTful API,
- SPA (Single Page Application),
- možnosť jednoduchého rozšírenia o ďalšie komponenty a funkcionality.

5 POUŽITÉ TECHNOLÓGIE

V tejto kapitole budú predstavené technológie, ktoré som využívala pri implementácii zadania pre vytvorenie RESTovej služby pre prístup do databázy. Pre RESTovú službu ako takú, by mi stačilo implementovať BE aplikácie a funkčnosť otestovať napríklad v Postmanovi, ja som však na základe zadania pridala aj jednoduchý FE.

Zoznam technológií, ktoré som používala, je nasledovný:

- intelliJ IDEA,
- komponenty MEAN stack,
- npm,
- angular CLI,
- postman.



Obrázok 11: Architektúra MEAN stack.[26]

[26] Zdroj: https://image.slidesharecdn.com/meanstackslideshare-160617140930/95/building-modern-web-apps-with-mean-stack-12-638.jpg?cb=1466263773 [cit. 2017-05-13].

5.1 Použité technológie pre vývoj a testovanie

5.1.1 Vývojové prostredie IntelliJ IDEA

IntelliJ IDEA je jedným z produktov firmy JetBrains. Je dostupné v dvoch edíciách a to Ultimate a Community. Ultimate edícia je platenou verziou Intellij IDEA.

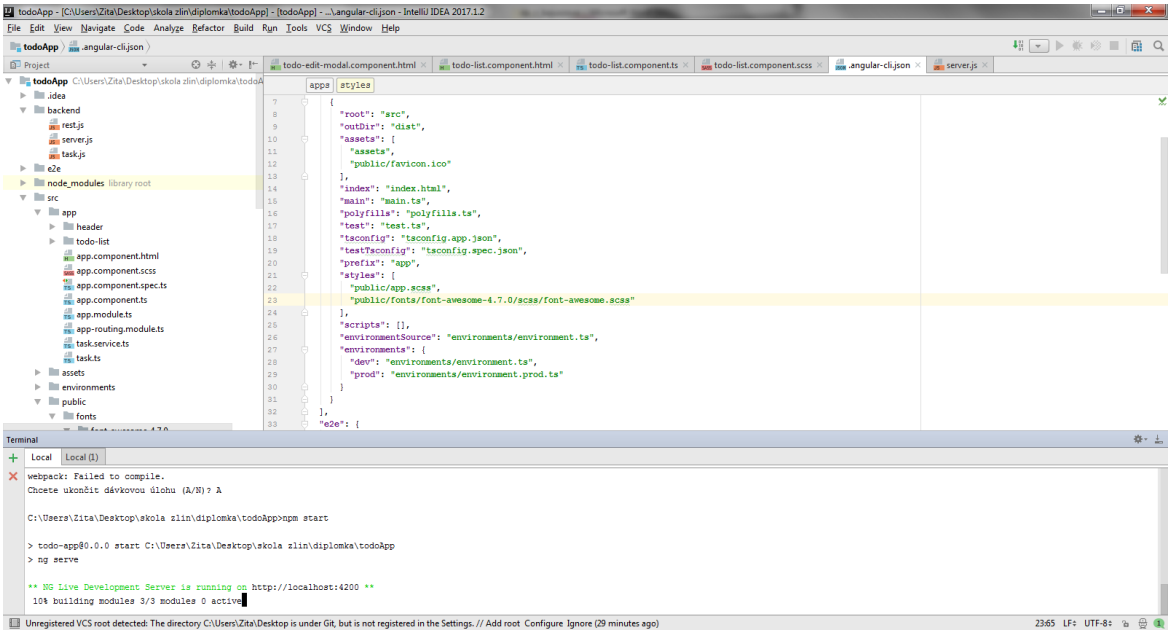
	Ultimate For web and enterprise development	**Community** For JVM and Android development
License	Commercial	Open-source, Apache 2.0
Java, Kotlin, Groovy, Scala	✓	✓
Android	✓	✓
Maven, Gradle, SBT	✓	✓
Git, SVN, Mercurial, CVS	✓	✓
Detecting Duplicates	✓	
Perforce, ClearCase, TFS	✓	
JavaScript, TypeScript	✓	
Java EE, Spring, GWT, Vaadin, Play, Grails, Other Frameworks	✓	
Database Tools, SQL	✓	

Obrázok 12: Porovnanie Ultimate a Community verzie pre Intellij IDEA

V rámci vývoja aplikácie som používala študentskú Ultimate verziu vývojového prostredia, keďže Community edícia neobsahuje podporu TypeScriptu.

Okrem podpory TypeScriptu ponúka toto IDE už v základnej verzii, bez potreby inštalácie ďalších pluginov, množstvo funkcií pre prácu s kódom, ako napríklad:

- Indexovanie zdrojového kódu – vďaka nemu poskytuje IDE relevantné návrhy na dokončenie kódu v každom kontexte a spoľahlivé nástroje na refaktorovanie,
- Integrované nástroje na verzionovanie,
- Inteligentné návrhy pri ponuke na dokončenie kódu - zatiaľ čo základné dokončenie naznačuje názvy tried, metód, polí a kľúčových slov v rámci rozsahu viditeľnosti, inteligentné návrhy pri ponuke na dokončenie kódu naznačujú iba tie typy, ktoré sa očakávajú v aktuálnom kontexte.
- Podpora pre veľké množstvo programovacích jazykov a frameworkov,
- Module dependency diagram a mnoho ďalších užitočných funkcionalít.



Obrázok 13: Vývojové prostredie IntelliJ IDEA

5.1.2 npm[27]

npm je balíčkovací systém, pomocou ktorého je možné inštalovať balíčky (moduly) v Node.js, pričom balíček, alebo modul, môže byť jednoduchý skript, alebo kompletný MVC framework. Balíčky rozdeľujeme do dvoch skupín. Do prvej skupiny patria tie, ktoré sú dodávané ako súčasť Node.js, do druhej naopak tie, ktoré sa inštalujú podľa potreby.

Všetky moduly je možné nájsť na webovej stránke https://www.npmjs.com/. S npm sa pracuje cez príkazový riadok alebo terminál v samotnom vývojom prostredí. Moduly sa inštalujú príkazom npm install. Ak si chceme napríklad nainštalovať angular CLI, použijeme príkaz

```
npm install -g @angular/cli
```

(-g argument znamená, že balíček sa nainštaluje globálne)[28].

Všetky nainštalované balíčky sa v štruktúre projektu nachádzajú v súbore node_modules. Tiež sú uvedené v package.json v časti dependencies, resp. devDependencies.

[27] Oficiálna stránka npm https://www.npmjs.com/
[28] Ďalšie používané argumenty je možné nájsť na webovej stránke https://docs.npmjs.com/cli/install

5.1.3 Postman

Postman slúži na testovanie REST API a je možné ho nainštalovať ako rozšírenie prehliadača Chrome.

Podrobný popis testovania pomocou Postmana bude uvedený v kapitole 6 NÁVRH APLIKÁCIE A REALIZÁCIA, konkrétne v časti 6.4.3 Popis testovania API.

5.2 FE aplikácie

5.2.1 Angular[29]

Angular je client-side webový framework napísaný v jazyku JavaScript. Aktuálne je vonku verzia Angular4. Využíva návrhový vzor MVC, prípadne vzor Model-View-ViewModel, vďaka čomu sa väčšina aplikačnej logiky nachádza v modely, resp. controllery. Controller alebo model sa následne vloží do $scope view šablony, s ktorou je možné manipulovať. Angular využíva vlastnú implementáciu sccope, vďaka ktorej si drží referencie na aktuálne premenné tiež časti DOM-u, s ktorým sa práve pracuje.

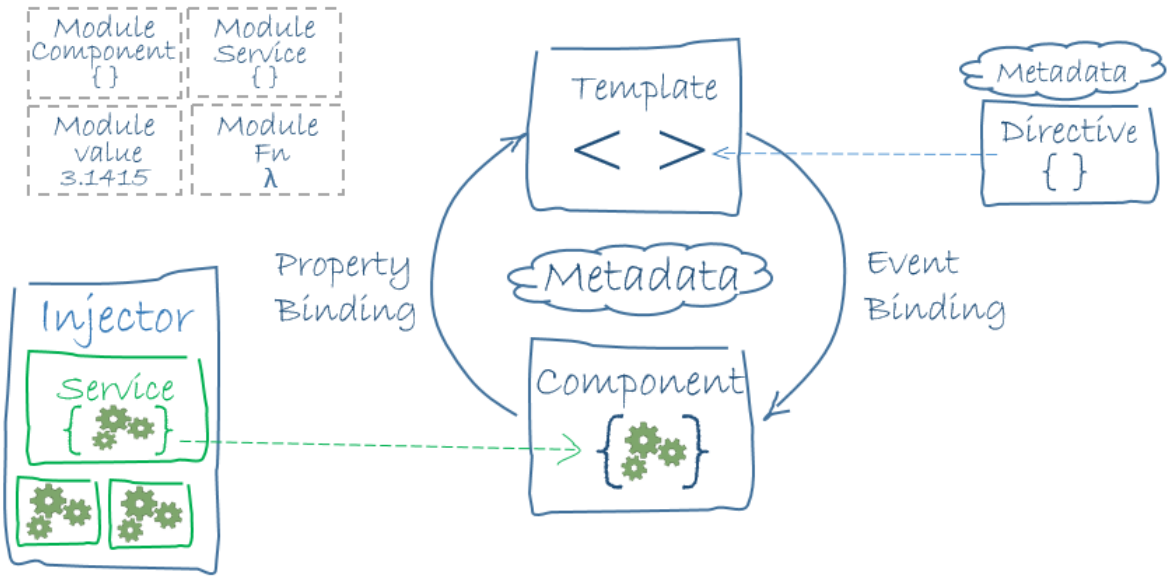
Na tvorbu šablón sa využíva jazyk HTML, ktorý ale nepotrebuje explicitné obnovenie DOM-u (Document Object Model), pretože Angular dokáže pomocou tzv. dirty-checkingu sledovať akcie užívateľa, prehliadača a zmeny v modeli. Automaticky rozpozná stav, v ktorom sa majú zmenené prvky obnoviť, a ktorých šablón sa obnova týka.

Z pohľadu architektúry sa Angular skladá z ôsmych stavebných blokov, ktorými sú:

- Moduly
- Komponenty
- Templaty
- Metadáta
- Data binding
- Direktívy
- Servisy
- Dependency injection

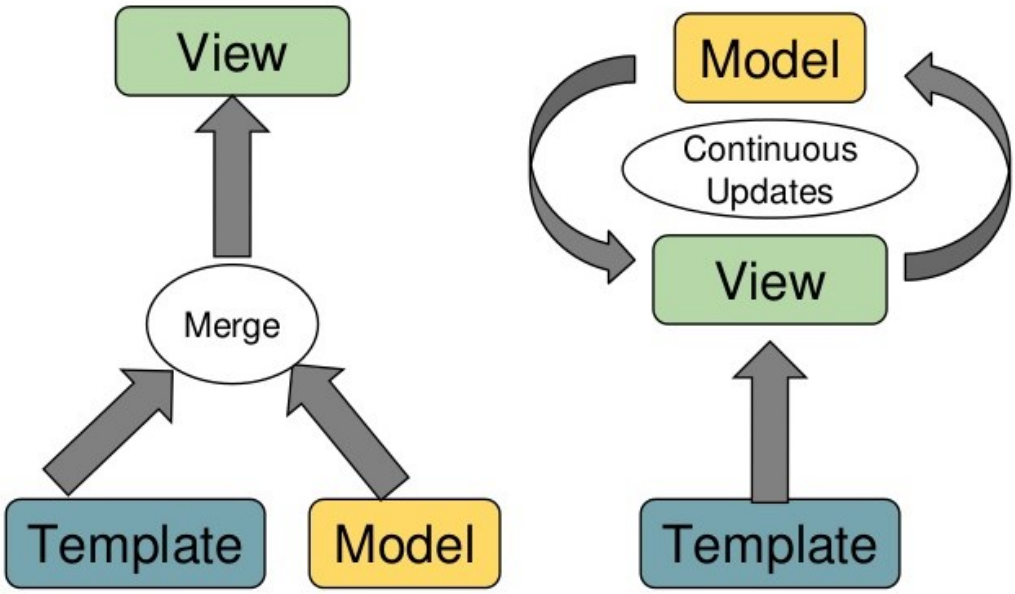
[29] Aktuálnu dokumentáciu je možné nájsť na https://angular.io/docs/ts/latest/guide/

Ich vzájomný vzťah zobrazuje nasledujúca schéma:



Obrázok 14: Architektúra Anularu4[30]

Zaujímavosťou Angularu je two–way data binding, ktorý rieši synchronizáciu stavov medzi modelom a view. Túto funkcionalitu ponúkajú aj iné frameworky, Angular ju má ale výborne automatizovanú.



Obrázok 15: Angular - one vs. two way data binding[31]

[30] Zdroj: https://angular.io/resources/images/devguide/architecture/overview2.png [cit. 2017-05-06].

Angular je možné rozširovať pomocou npm balíčkov o ďalšie moduly. Ja som si pre potreby vývoja doinštalovala balíčky angular CLI, bootstrap4 a ng-bootrap .

Angular CLI[32]

Angular CLI je nástroj na inicializáciu, vývoj a údržbu Angular aplikácií.

Inštaluje sa pomocou príkazového riadku, podľa aktuálnej dokumentácie prostredníctvom príkazu

```
npm install -g @angular/cli
```

Následne je možné nový vytvoriť a spustiť nový projekt pomocou sady príkazov:

```
ng new my-project
cd new-project
ng serve
```

Ďalšie príkazy:

- ng generate – generuje špecifikovanú štruktúru, ako napríklad enum, pipe alebo service,
- ng lint – skontroluje váš kód prostredníctvom tslint,
- ng test – umožní spustiť unit testy,
- ng e2e – vykonáva testy typu end-to-end,
- ng build - skompiluje aplikáciu do výstupného adresára,
- ng get/ng set – pomocou tohto príkazu získať alebo nastaviť hodnotu z konfigurácie,
- ng doc – otvorí oficiálnu Angular dokumentáciu,
- ng eject – vypublikuje aplikáciu a vygeneruje správnu webpack konfiguráciu a skripty,
- ng xi18n - extrahuje správy i18n zo šablón.

---

[31] Zdroj: https://image.slidesharecdn.com/qew2fe1qfouq3ez0irrw-signature-a5e16f8a5f9bf5c06bfd992151090953475efa37da75f9eed6674474188d6d90-poli-141218052940-conversion-gate02/95/angular-js-10-638.jpg?cb=1418892818 [cit. 2017-05-06].

[32] https://github.com/angular/angular-cli/wiki

5.2.2 Bootstrap/Sass

Bootstrap je sada nástrojov pre tvorbu webu a webových aplikácií. Obsahuje sadu štýlov a JavaScript rozšírenia pre úpravu vzhľadu aplikácie. Aktuálne je k dispozícii verzia Bootstrap4, ktorú som využila v rámci vývoja. Dôvodom, prečo som si vybrala túto verziu napriek tomu, že sa jedná o alpha verziu je tá, že natívne podporuje preprocesor Sass a flexbox pre tvorbu responzívneho dizajnu.

Inštaluje sa ako npm balíček pomocou príkazu:

```
npm install --save bootstrap@4.0.0-alpha.6
```

Balíček sa v štruktúre projektu umiestni do node_modules.

Pre potreby ďalšieho vývoja sa neupravujú tieto súbory, aby sa v prípade stiahnutia vyššej verzie neprepísali, ale v štruktúre projektu sa, napr. v súbore public, vytvoria potrebné kópie, s ktorými sa ďalej pracuje.

ng-bootstrap

ng-bootstrap obsahuje komponenty špeciálne vyvinuté pre ekosystém Angularu. Komponenty používajú Boostrap 4 CSS a neobsahujú žiadnu závislosť na JavaScriptových knižniciach tretích strán. Pre potreby aplikácie som použila komponent Modal.

Prerekvizity pre ng-boostrap sú:

- Angular (aktuálne verzia 4 alebo vyššie),
- Bootstrap CSS (testované s verziou 4.0.0-alpha.6)

Podporované prehliadače zodpovedajú podporovaným prehliadačom pre Angular a Bootstrap.

ng-bootstrap je dostupný ako npm balíček. Inštaluje sa pomocou príkazu:

```
npm install --save @ng-bootstrap/ng-bootstrap
```

Po inštalácií je potrebné importovať ho do hlavného modulu:

```
import {NgbModule} from '@ng-bootstrap/ng-bootstrap';
@NgModule({
  declarations: [AppComponent, ...],
  imports: [NgbModule.forRoot(), ...],
  bootstrap: [AppComponent]
})
```

```
export class AppModule {}
```

Pre ostatné moduly potom stačí už iba jednoduchý import imports: [NgbModule, ...].

5.3 BE aplikácie

5.3.1 Node.js

Node.js je cross-platformové open-source run-time prostredie, ktoré umožňuje aplikáciu JavaScriptu na strane servera. Používa sa napríklad na vytváranie dynamických webových stránok. Výhodou je, že obsah sa generuje na strane servera a nezaťažuje klienta. Skladá sa z V8 JavaSCript engine od spoločnosti Google a niekoľkých štandardných knižníc.

Medzi základné vlastnosti Node.js patria:

- Vysoká škálovateľnosť,
- Každá funkcia je asynchrónna – všetko, čo by blokovalo vlákno, sa vykonáva na pozadí. Pre implementáciu to znamená, že je potrebné určiť callback funkciu, ktorá sa zavolá, keď sa funkcia na pozadí skončí.
- Bez vyrovnávacej pamäte.

Node.js je vhodný pre vývoj I/O aplikácií, data streaming, real time aplikácie, aplikácie založené na báze JSON API a single page aplikácie.

Príklad servera Hello World:

```
var http = require('http');
http.createServer(function (request, response) {
  response.writeHead(200, {'Content - Type': 'text / plain'
  });
  response.end('Hello World');
}).listen(8000);
console.log('Server running at http: //localhost:8000/');
```

5.3.2 Express

Express je framework postavený na Node.js. Podrobnejšie sme si ho prestavili v kapitole 3.

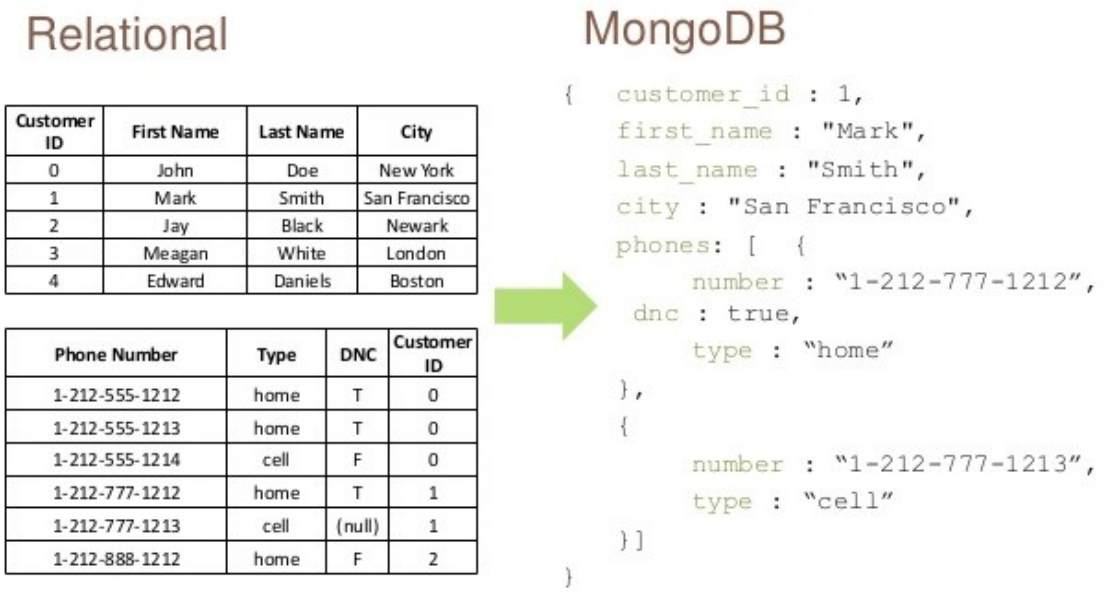
Dôvodom, prečo som ho uprednostnila pred ostatnými frameworkami je jeho jednoduchosť, podpora zo strany webovej komunity, množstvo dokumentácie a príkladov.

5.4 Databáza

Pre prácu s databázou som použila MondoDB, Mongoose a Mongoclient.

5.4.1 MongoDB

MongoDB[33] je NoSQL open source databáza, ktorá namiesto relačných databáz používa dokumenty podobné formátu JSON (nazývajú sa BSON) a dynamické dátové schémy.



Customer ID	First Name	Last Name	City
0	John	Doe	New York
1	Mark	Smith	San Francisco
2	Jay	Black	Newark
3	Meagan	White	London
4	Edward	Daniels	Boston

Phone Number	Type	DNC	Customer ID
1-212-555-1212	home	T	0
1-212-555-1213	home	T	0
1-212-555-1214	cell	F	0
1-212-777-1212	home	T	1
1-212-777-1213	cell	(null)	1
1-212-888-1212	home	F	2

Obrázok 16: Rozdiel medzi relačnou a dátovo orientovanou databázou.[34]

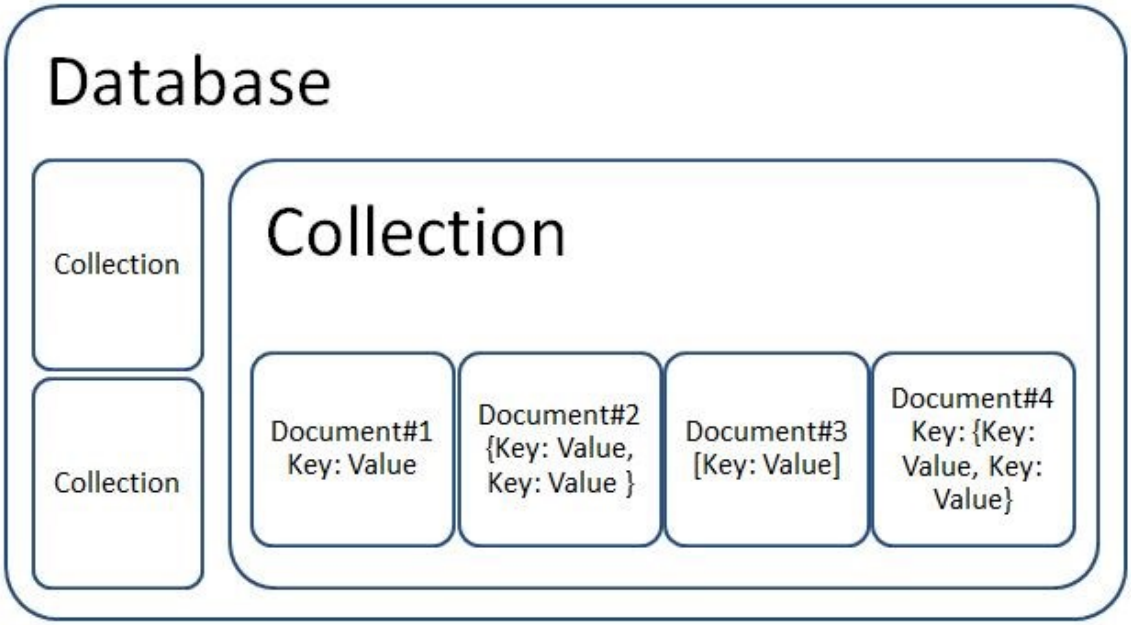
Dokumenty ukladá do kolekcií, čo je ekvivalent tabuľky v relačných databázach. Každá kolekcia musí mať svoje svoje vlastné unikátne _id, každý dokument môže obsahovať JSON, XML, YAML alebo dokonca aj dokument vo formáte Word. Kvôli jej silnej orientácii na dokumenty sa označuje ako databáza orientovaná na dokumenty.

Orientácia na dokumenty

- Business subjekty sú ukladané v minimálnom množstve dokumentov. V prípade mojej aplikácie mi postačuje jeden dokument, resp. kolekcia, ktorá si drží informácie týkajúce sa úlohy, vrátane statusu.
- Minimálne množstvo dokumentov redukuje potrebu spájania tabuliek, čo zrýchľuje aplikáciu

[33] Dokumentácia a postup k práci s MongoDB je dostupná na https://docs.mongodb.com/getting-started/shell/.
[34] Zdroj: https://image.slidesharecdn.com/howinsuranceusemongo2-140731131220-phpapp01/95/how-insurance-companies-use-mongodb-10-638.jpg?cb=1406812413 [cit. 2017-05-06].

- Dynamická schéma uľahčuje polymorfizmus



Obrázok 17: Dátový model databázy orientovanej na dokumenty.[35]

Poskytuje vysoký výkon, dostupnosť a ľahkú škálovateľnosť.

Vysoký výkon

- Akékoľvek pole dokumtentu môže byť indexované, dostupné sú aj sekundárne indexy

Vysoká dostupnosť

- Je k dispozícií pomocou sady replík. Každá takáto sada obsahuje dve alebo viac kópií dát. Všetky funkcie realizuje primárna replika. Sekundárne repliky udržiavajú kópiu dát primárnej repliky. Keď primárna replika zlyhá, automaticky sa vyberie za ňu náhrada.

Ľahká škálovateľnosť

- MongoDB sa škáluje horizontálne pomocou shardingu
- Môže bežať na viacerých serveroch, tzn. že konzistentné čítanie môže byť distribuované cez replikované servery.

[35] Zdroj: https://dzone.com/storage/temp/966093-03-documentdbv1.jpg [cit. 2017-05-13].

5.4.2 Mongoose[36]

Mongoose slúži na modelovanie objektov mongodb pre node.js. MongoDB je "schema less", tzn. že v prípade potreby implementácie štruktúrovanejšej databázy s využitím všetkých funkcionalít MongoDB potrebujete ďalší nástroj. Jedným z riešení pre ODM (Mapovanie objektových dát) je práve Mongoose.

Napríklad, uvedený insert vloží do kolekcie s názvom používateľov záznam:

```
db.users.insert ({meno: 'Zita', pohlavie: 'žena'});
```

Hneď na to môžeme spustiť ďalší insert:

```
db.users.insert ({meno: 'Zita', pohlavie: 'žena', heslo: 'hes-
lo123'});
```

MongoDB nebude vadiť zmena počtu stĺpcov, čo je síce veľmi flexibilné, ale nevýhodné, ak chcete, aby vaše dáta boli organizované a štruktúrované. Miesto toho, aby to bolo potrebné pracne ošetrovať na strane servera, sa použije Mongoose so zabudovaným overovaním dotazov a ďalšími užitočnými funkcionalitami.

Prerekvizity pre Mongoose sú:

- MongoDB
- Node.js

Mongoose je dostupný ako npm balíček. Inštaluje sa pomocou príkazu:

```
npm install --save mongoose
```

Ukážka krokov potrebných pre vytvorenie spojenia s databázou na backende:

1. Mongoose je potrebné zahrnúť do projektu a otvoriť spojenie s testovacou, resp. vývojovou, databázou na lokálne bežiacej inštancii MongoDB.

   Ukážka kódu aplikácie, súbor rest.js:

   ```
   var express = require('express');
   var mongoose = require('mongoose');
   var Task = require('./task');
   ```

[36] Dokumentácia je dostupná na stránke http://mongoosejs.com/index.html [cit. 2017-05-13].

```
//connect to mongodb todoApp Databasen
mongoose.connect('mongodb://127.0.0.1:27017/todoApp');
```

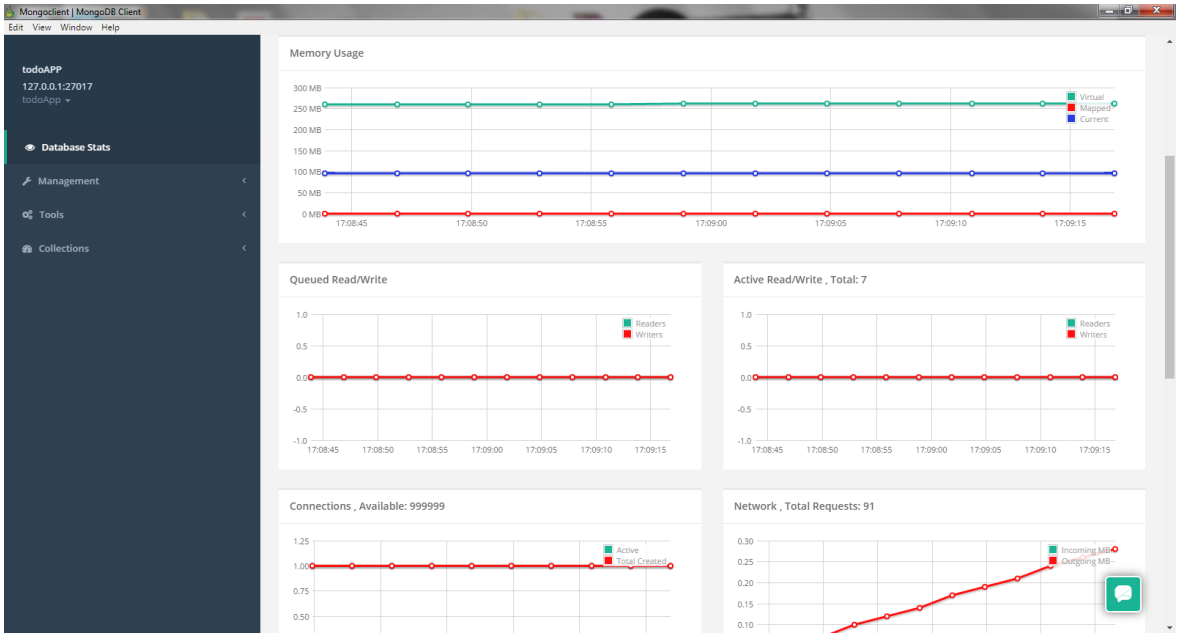
2. V Mongoose je všetko odvodené od schémy. Definícia schémy v aplikácii sa nachádza v súbore task.js:

```
// Schema
var TaskSchema = new mongoose.Schema({
  title: String,
  description: String,
  deadline: Date,
  status: Boolean
});
```

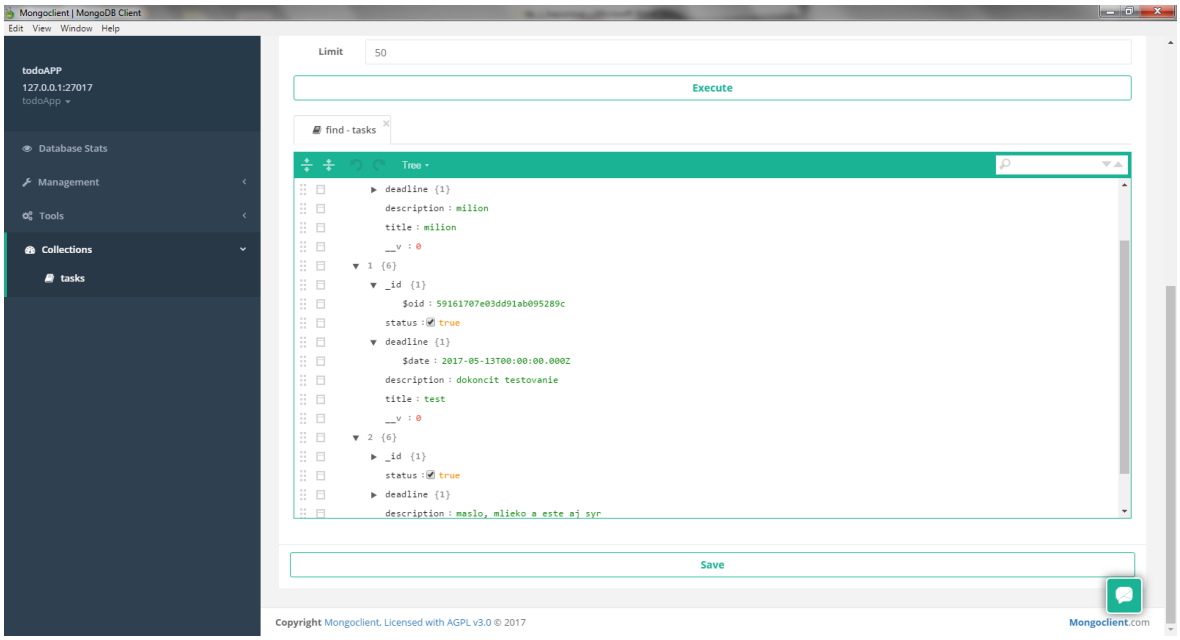
5.4.3 Mongoclient

Mongoclient je jedným z dostupných GUI rozhraní pre správu mongoDB. Je k dispozícii aj ako open source. V tejto verzii poskytuje nasledujúce funkcionality:

- LDAP, GSSAPI, X509 autentifikáciu
- Živý monitoring
- Mongodb shell
- SSH tunelovanie
- In-place update
- Mongoimport and CSV/JSON export
- Index manažment
- Správu užívateľov
- Správu adresárov
- Drag & drop nástroj
- Rozšírenie JSON-u pre dotazy
- Statické zobrazenia viacerých pohľadov
- Automatické dokončenie poľa
- Analyzátor schémy



Obrázok 18: Mongoclient – úvodná obrazovka so štatistickými údajmi



Obrázok 19: Mongoclient – obrazovka zobrazujúca kolekciu pre tasks

5.5 Dátový formát pre reprezentáciu dát

Ako dátový formát pre reprezentáciu dát som použila JSON. O JSON-e píšem podrobnejšie v kapitole 2 RESTOVÉ SLUŽBY, kontrétne v časti 2.1.3.2 JSON.

Príklad výstupu pre getTasks v konzole pre prehliadač Chrome:

```
Object
deadline:"2017-05-13T00:00:00.000Z"
```

```
description:"maslo, mlieko a este aj syr"
status:true
title:"nakupit"
__v:0
_id:"59161a00e03dd91ab095289d"
```

6 NÁVRH APLIKÁCIE A REALIZÁCIA

Samotná aplikácia slúži iba ako frontendové rozhranie na testovanie RESTového API.

Pred samotným spracovaním tejto témy som sa rozhodovala, či navrhnuté API otestujem iba v niektorom s testovacích nástrojov, ako napríklad Postman alebo curl. Nakoniec som sa rozhodla naprogramovať pre tento účel jednoduchú aplikáciu, pomocou ktorej bude možné v užívateľsky príjemnom prostredí otestovať všetky operácie CRUD-u.

Z toho dôvodu sa nejedná o zložitú aplikáciu, tiež nie sú na frontende aplikované funkcionality ako validácia polí a pod.

Aplikácia predstavuje jednoduché webové rozhranie na správu úloh. Užívateľ si prostedníctvom nej môže vytvárať a manažovať svoje úlohy.

Architektonicky je postavená ako SPA (single page application), tzn. že server používa iba ako zdroj a úložisko dát. Dáta sú kompletne vykreslované frontendom. Hlavnou výhodou SPA je rýchla odozva po prvotnom načítaní a inicializácii sriptov. Zo servera sa už potom nesťahuje celý HTML kód, ale len dáta, ktoré sa menia. Ďalšou výhodou je pomerne jednoduché vytvorenie, keďže sa logika aplikuje len na jednej stránke na strane klienta. SPA aplikácie nepotrebujú ani serverovú technológiu pre vytváranie stránok, vystačia si so statickým http serverom, ktorý posiela HTML, CSS a JS súbory klientovi.

Štruktúra aplikácie je, v súlade s vyššie uvedeným, rozdelená na frontendovú (FE) a backendovú časť (BE). BE obasahuje implementáciu serverovej logiky (server.js), schému pre databázu (task.js) a API pre prístup do databázy (rest.js). FE obsahuje komponenty pre zobrazovanie dát a potrebné servisy.

Základná štruktúra:

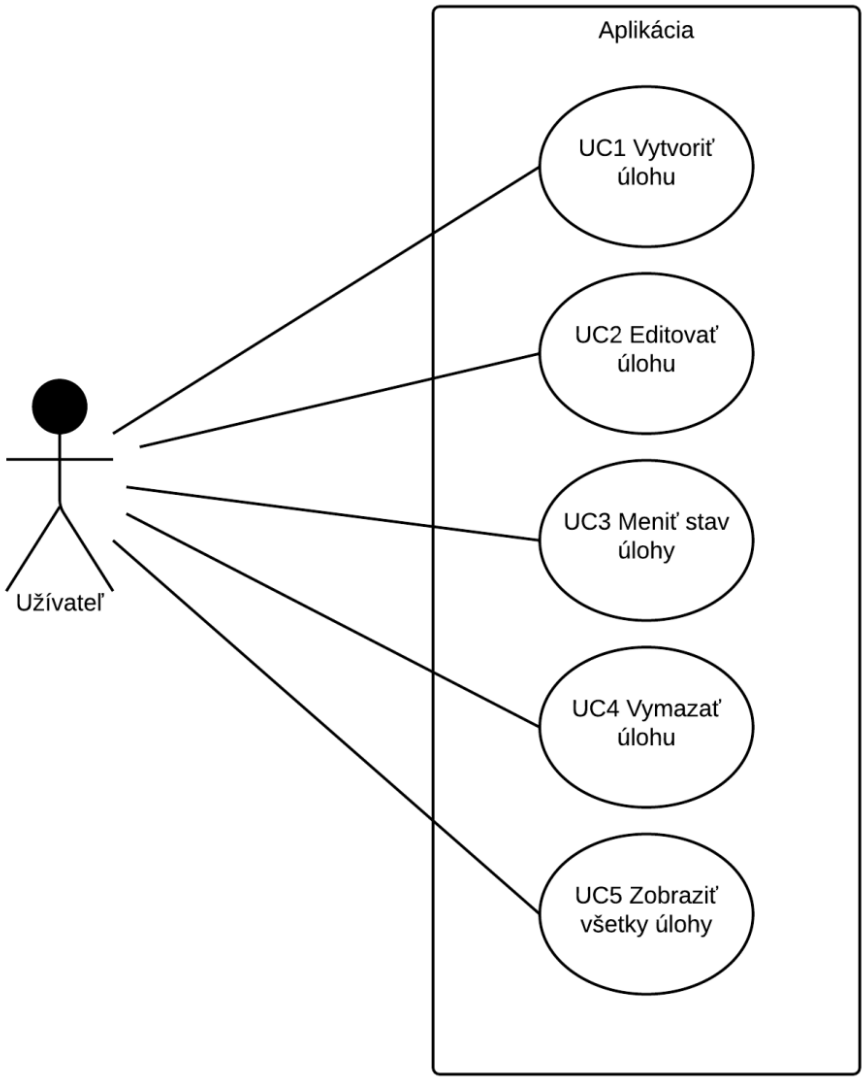
- .angular-cli.json
- .editorconfig
- .gitignore
- .idea
- backend (zložka obsahuje súbory pre BE aplikácie)
- e2e
- karma.conf.js
- node_modules (zložka obsahuje všetky npm balíčky)
- package.json

- protractor.conf.js
- README.md
- src (zložka obsahuje súbory pre FE aplikácie)
- tsconfig.json
- tslint.json

6.1 Prípady užitia

Pri popise prípadov užitia budem vychádzať z UC diagramu, v ktorom je zachytený aktér a jeho vzťah k jednotlivých funkcionalitám. Pre tvorbu diagramov bola použitá webová aplikácia Lucidchart.

6.1.1 UC diagram



Obrázok 20: Diagram prípadov užitia

6.1.2 Aktéri

Systém má iba jeden typ aktéra a tým je užívateľ. Vstup do aplikácie nie je nijako obmedzený, nie je požadované prihlásenie, ani iný spôsob autentifikácie.

Užívateľ má oprávnenie na všetky funkcionality, ktoré aplikácia poskytuje.

6.1.3 UC špecifikácie

UC1 Vytvoriť úlohu

Krátky popis

Táto funkcionalita umožňuje aktérovi pridať novú úlohu do zoznamu úloh (Create/Post).

Aktéri

Užívateľ

Podmienky pre spustenie

Užívateľ sa musí nachádzať na úvodnej stránke aplikácie.

Základný tok

Aplikácia zobrazí úvodnú stránku, na ktorej sa nachádza tlačidlo pre zadanie novej úlohy.

Po stlačení sa zobrazí modálne okno so vstupnými poľami pre zadanie úlohy, jej popis a deadline.

Užívateľ vyplní polia a stlačí tlačidlo Uložiť.

Systém uloží novú úlohu a pridá ju do zoznamu.

Alternatívny tok

Neexistuje

Podmienky pre dokončenie

Nová úloha bude korektne uložená do databázy.

UC2 Editovať úlohu

Krátky popis

Táto funkcionalita umožňuje aktérovi editovať už uloženú úlohu, ktorá je aktívna a nachádza sa v zozname úloh (Update/Put).

Aktéri

Užívateľ

Podmienky pre spustenie

Užívateľ sa musí nachádzať na úvodnej stránke aplikácie, kde je zobrazený zoznam zadaných úloh.

Základný tok

Aplikácia zobrazí úvodnú stránku, na ktorej sa nachádzajú všetky zadané úlohy.

Pri každej úlohe sa nachádzajú ikony so znakom pre vymazanie, editovanie a označenie úlohy ako splnenej.

Po stlačení ikony pre editovanie bude možné úlohu editovať v modálnom okne a opäť uložiť.

Systém uloží nové znenie úlohy.

Alternatívny tok

Neexistuje

Podmienky pre dokončenie

Editovaná úloha bude korektne uložená do databázy.

UC2 Meniť stav úlohy

Krátky popis

Táto funkcionalita umožňuje aktérovi meniť stav už uloženej úlohy. Úloha môže nadobúdať dva stavy a to splnená alebo nesplnená. V databáze prezentuje tento stav dátový typ boolean. (Update/Put).

Aktéri

Užívateľ

Podmienky pre spustenie

Užívateľ sa musí nachádzať na úvodnej stránke aplikácie, kde je zobrazený zoznam zadaných úloh.

Základný tok

Aplikácia zobrazí úvodnú stránku, na ktorej sa nachádzajú všetky zadané úlohy.

Pri každej úlohe sa nachádzajú ikony so znakom pre vymazanie, editovanie a označenie úlohy ako splnenej.

Po označení úlohy ako splnenej sa úloha presunie do zoznamu splnených úloh.

Systém uloží nový status úlohy.

Alternatívny tok

Užívateľ sa nachádza v záložke splnených úloh.

Pomocou refresh ikony môže zmeniť stav úlohy a označiť ju opäť ako nesplnenú.

Podmienky pre dokončenie

Systém si uloží stav, v akom sa úloha nachádza.

UC4 Vymazať úlohu

Krátky popis

Táto funkcionalita umožňuje aktérovi vymazať úlohu, ktorá je aktívna a nachádza sa v zozname úloh (Delete/Delete).

Aktéri

Užívateľ

Podmienky pre spustenie

Užívateľ sa musí nachádzať na úvodnej stránke aplikácie, kde je zobrazený zoznam zadaných úloh.

Základný tok

Aplikácia zobrazí úvodnú stránku, na ktorej sa nachádzajú všetky zadané úlohy.

Pri každej úlohe sa nachádzajú ikony so znakom pre vymazanie, editovanie a označenie úlohy ako splnenej.

Po stlačení ikony pre mazanie bude vybraná úloha vymazaná a nebude sa k nej možné vrátiť.

Systém vymaže úlohu.

Alternatívny tok

Neexistuje

Podmienky pre dokončenie

Úloha bude korektne vymazaná z databázy.

UC5 Zobraziť všetky úlohy

Krátky popis

Táto funkcionalita umožňuje aktérovi zobraziť všetky zadané úlohy (Read/Get).

Aktéri

Užívateľ

Podmienky pre spustenie

Užívateľ sa musí nachádzať na úvodnej stránke aplikácie.

K dispozícii sú dva zoznamy. Jeden pre splnené a druhý pre nesplnené úlohy.

Prepínať sa medzi nimi je možné pomocou záložiek, ktoré sa nachádzajú na úvodnej stránke.

Základný tok

Aplikácia zobrazí úvodnú stránku, na ktorej sa nachádzajú všetky úlohy.

Alternatívny tok

Neexistuje

Podmienky pre dokončenie

Na úvodnej stránke sú zobrazené všetky úlohy, ktoré sú korektne uložené v databáze.

6.2 Dátový model

Databázu aplikácie a spôsob jej fungovania som podrobnejšie opísala v kapitole 5 POUŽITÉ TECHNOLÓGIE, konkrétne v časti 5.4.2 Mongoose.

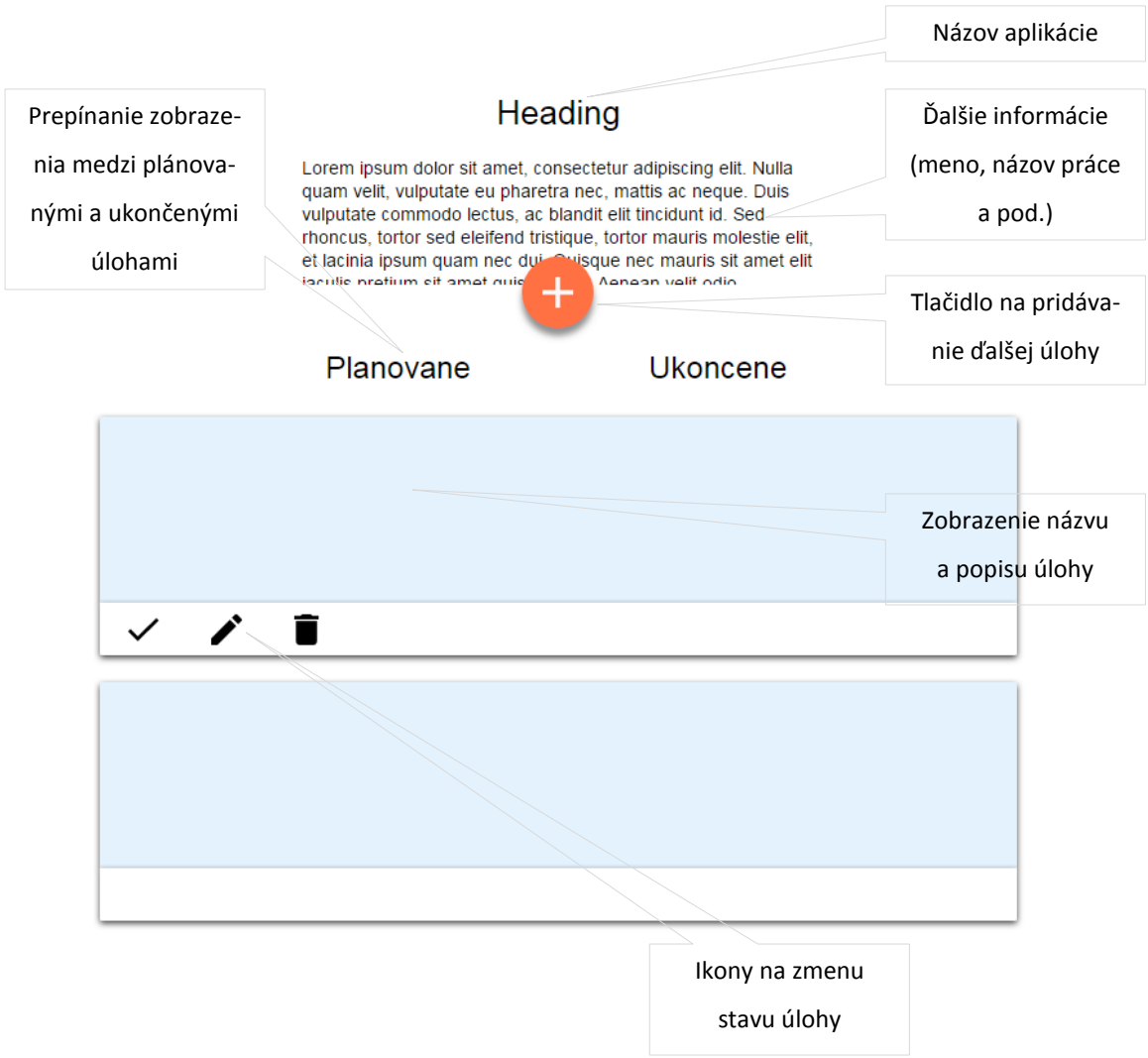
Dátový model aplikácie je jednoduchý, obsahuje len jednu kolekciu, ktorá si drží informácie o úlohe.

6.3 Popis obrazoviek

Aplikácia obsahuje jednu obrazovku s možnosťou prepínať sa medzi dvoma pohľadmi.

Nová úloha sa pridáva pomocou modálneho okna.

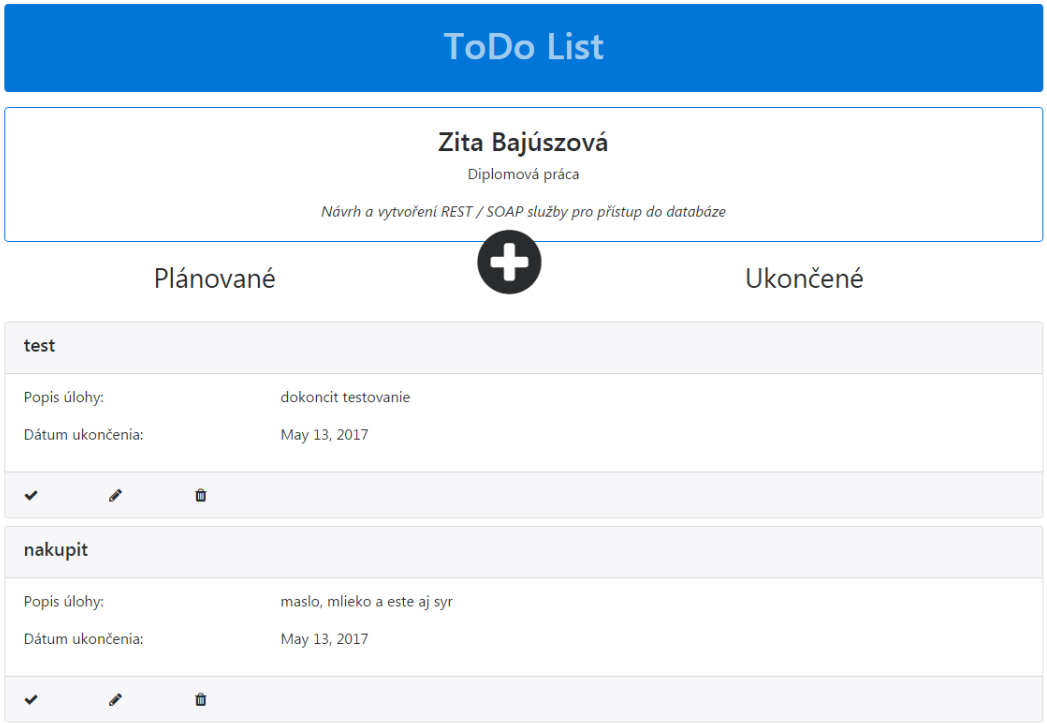
Wireframe obrazovky[37]



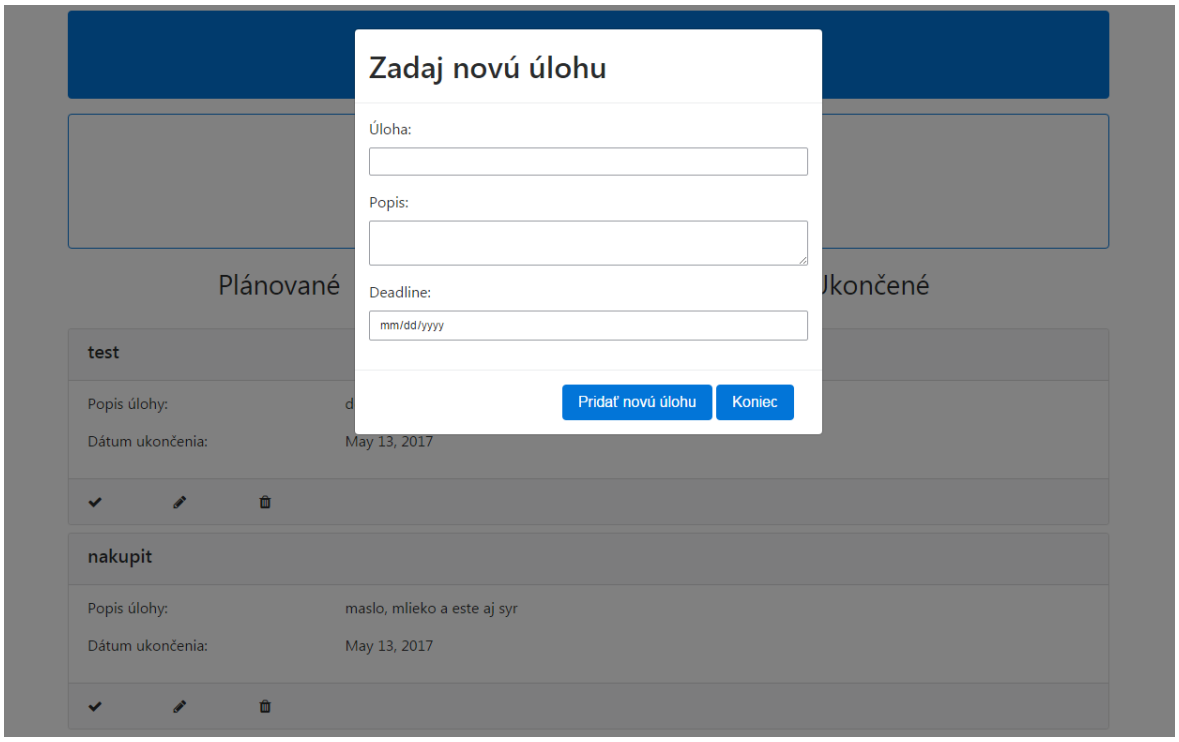
Obrázok 21: Wireframe základnej obrazovky

[37] Wireframe bol vypracovaný pomocou bezplatnej online verzie aplikácie Moqups. Dostupné na https://moqups.com.

Obrazovka aplikácie



Obrázok 22: Obrazovka aplikácie



Obrázok 23: Vloženie novej úlohy pomocou modálneho okna

6.4 Popis rozhrania pre BE služby

Rozhranie pre BE služby obsahuje implementáciu servera, schémy pre databázu a samotné API. O implementácii servera pomocou frameworku Express a schémy pre databázu som už písala vyššie.

V tejto podkapitole sa budem venovať primárne súboru rest.js, v ktorom je definované samotné API.

6.4.1 Metodika návrhu RESTful API

Pri návrhu REST API je potrebné dbať na niekoľko základných pravidiel. Na základe nich môžeme postup pri návrhu API rozdeliť do nasledujúcich krokov:

1. Definícia všeobecných vlastností služby
2. Definícia endpointov – obsahuje zoznam všetkých endpointov vrátane parametrov
3. Definícia metód – ku každému endpointu je potrebné uviesť metódy, aby bolo jasné, aké operácie bude možné realizovať nad daným zdrojom
4. Definícia dátových štruktúr – je potrebné uviesť typ a štruktúru dát, ktorými je možné v rámci endpointu pracovať. Pravidlom je, že v rámci endpointu nesmie dôjsť k nekonzistenciám naprieč metódami. Je to kvôli tomu, že výstup jednej metódy môže byť vstupom inej metódy v rámci toho istého endpointu.
5. Definícia chybových stavov

6.4.2 Manipulácia s dátami

Manupulácia s dátami je možná pomocou endpointov, ktoré sme si definovali.

/tasks

Metóda	**Popis**
POST	Pridanie novej úlohy
GET	Zobrazenie všetkých úloh

Tabuľka 7: Metódy pre endpoint /tasks

/tasks/:task_id

Metóda	Popis
DELETE	Vymazanie úlohy
PUT	Zmena statusu úlohy

Tabuľka 8: Metódy pre endpoint /tasks/:task_id

/tasks/edit/:task_id

Metóda	Popis
PUT	Editovanie úlohy

Tabuľka 9: Metódy pre endpoint /tasks/edit/:task_id

Smerovanie na jednotlivé endpointy je zadefinované v rest.js pomocou premenných, napríklad:

```
var taskEditRoute = router.route('/tasks/edit/:task_id');
```

Následne sú implementované jednotlivé metódy. Pre uvedený príklad je implementovaná metóda PUT, pomocou ktorej vieme editovať úlohy:

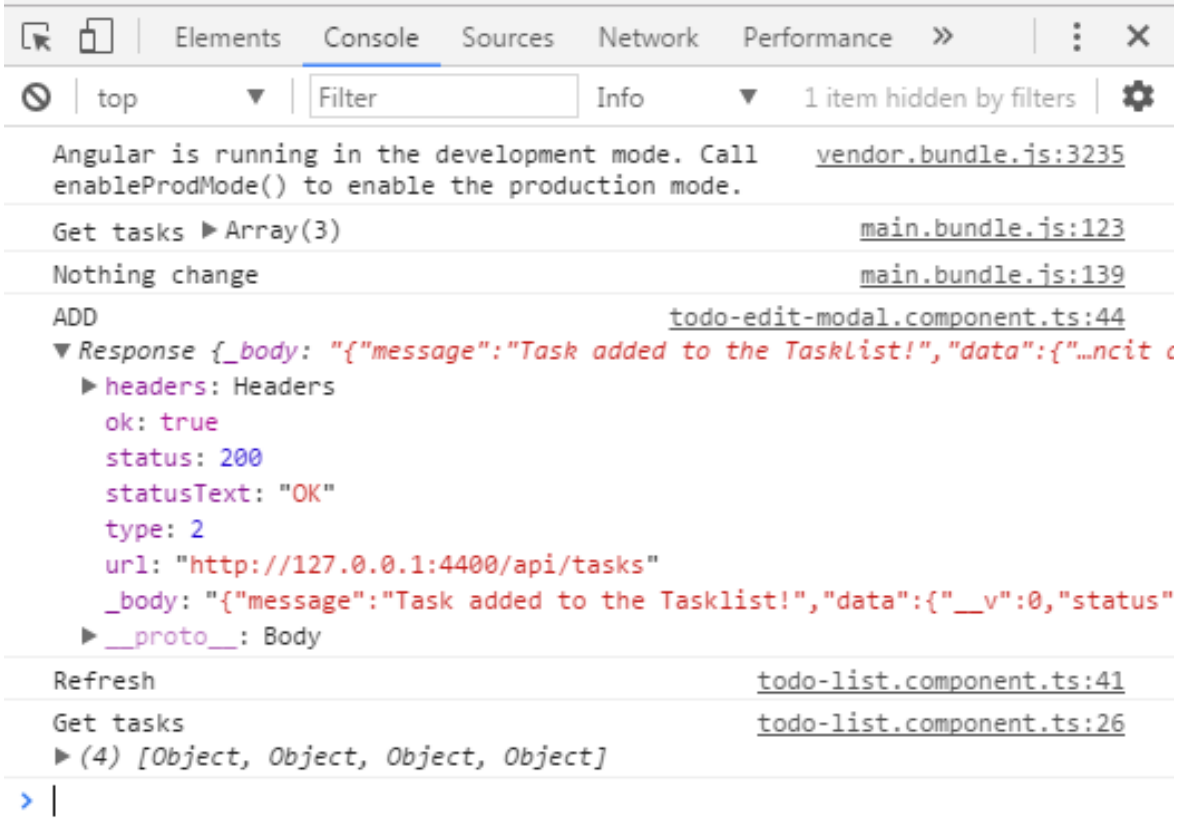
```
// Edit status
taskEditRoute.put(function (req, res) {
  // Find task and edit
  Task.findById(req.params.task_id, function (err, task) {
    if (err)
      res.send(err);
    task.title = req.body.title;
    task.description = req.body.description;
    task.deadline = req.body.deadline;
    task.save(function (err) {
      if (err)
        res.send(err);
      res.json(task);
    });
  });
});
```

Na samotnú manipuláciu s dátami ale BE implementácia nestačí. Na strane FE musí byť vytvorený súbor, ktorý bude obsahovať služby na odosielanie žiadostí na API server.
V prípade mojej implementácie je to typescriptový súbor task.service.ts.
Definícia CRUD operácie pre editovanie úlohy:

```
// Edit task
  editTask(task: Task): Observable<any> {
```

```
    const options = new RequestOptions({headers: this.headers,
method: 'put'});
    return this.http.put(`${this.tasksUrl}/edit/${task._id}`,
JSON.stringify(task), options);
}
```

Princíp fungovania aplikácie zobrazuje nasledujúci obrázok z konzoly prehliadača Chrome:



Obrázok 24: Ukážka fungovania aplikácie

Pri počiatočnej inicializácii sa zobrazí zoznam všetkých úloh, Nothig change znamená, že po vykonaní tejto funkcie nedôjde k refreshu stránky. Po pridaní novej úlohy (funkcia onSubmit nad formulárom v modálnom okne, konzolový záznam ADD), dôjde k refreshu stránky, aby sa opäť mohol načítať zoznam nových úloh, tento krát už aj s novo pridanou úlohou.

6.4.3 Popis testovania API

Testovanie softvéru je bežnou praxou mnohých programátorov. Dobrým nástrojom na rýchle testovanie API je Postman.

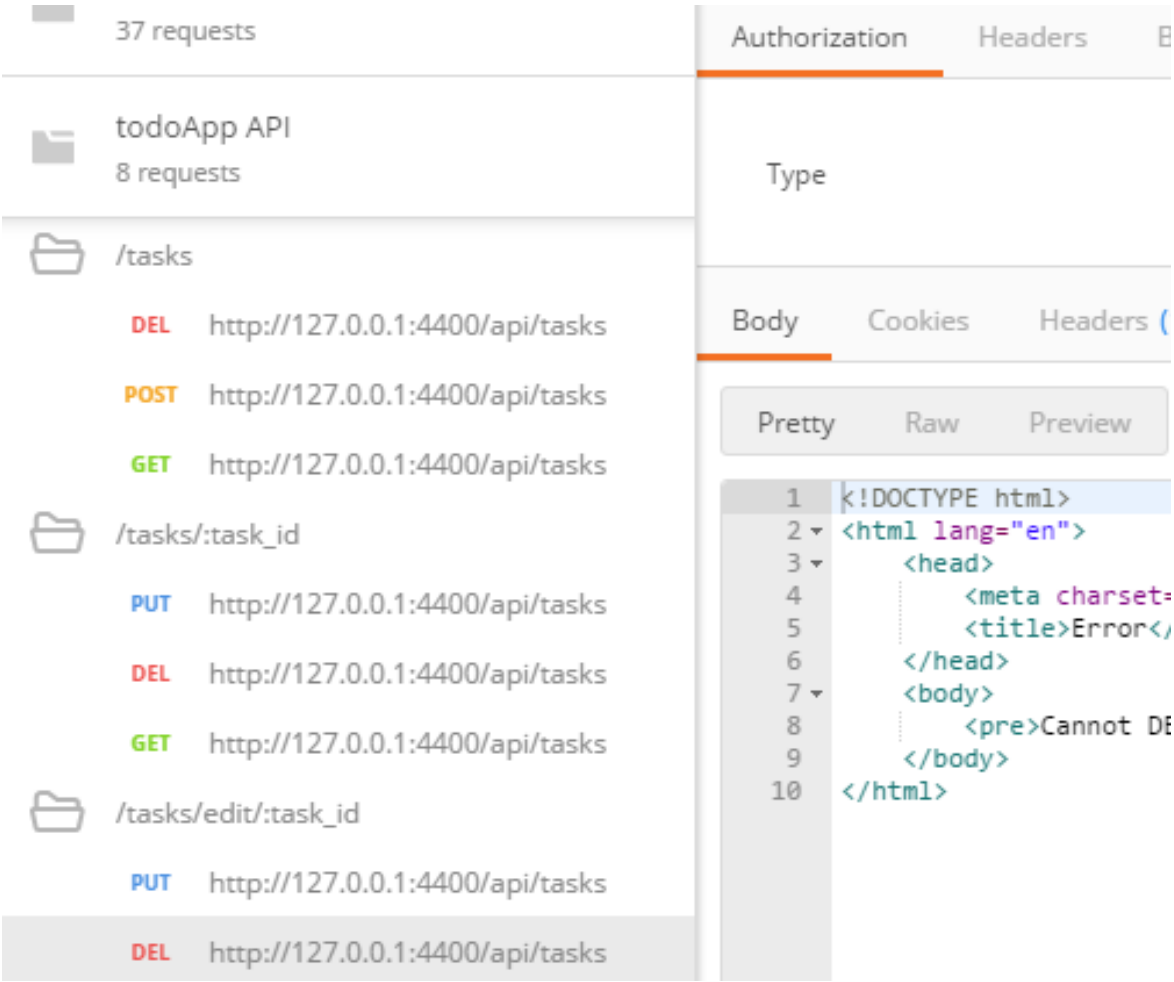
Zoznam jeho hlavných funkcionalít je nasledovný:

- Jednoduché testovanie akékoľvek rozhrania API, odpoveď je viditeľná ihneď,
- Možnosť vytvorenia kolekcie koncových bodov rozhrania API a ich volania vždy, keď chcete zobraziť odpovede.
- Tieto kolekcie je možné zdieľať po prihlásení medzi členmi tímu.
- Na písanie testov pomocou jazyka JavaScript je k dispozícii CLI.

Jeho inštalácia je veľmi jednoduchá. Aplikáciu je možné stiahnuť zo stránky Postman-a, alebo ju pridať ako rozšírenie prehliača Chrome.

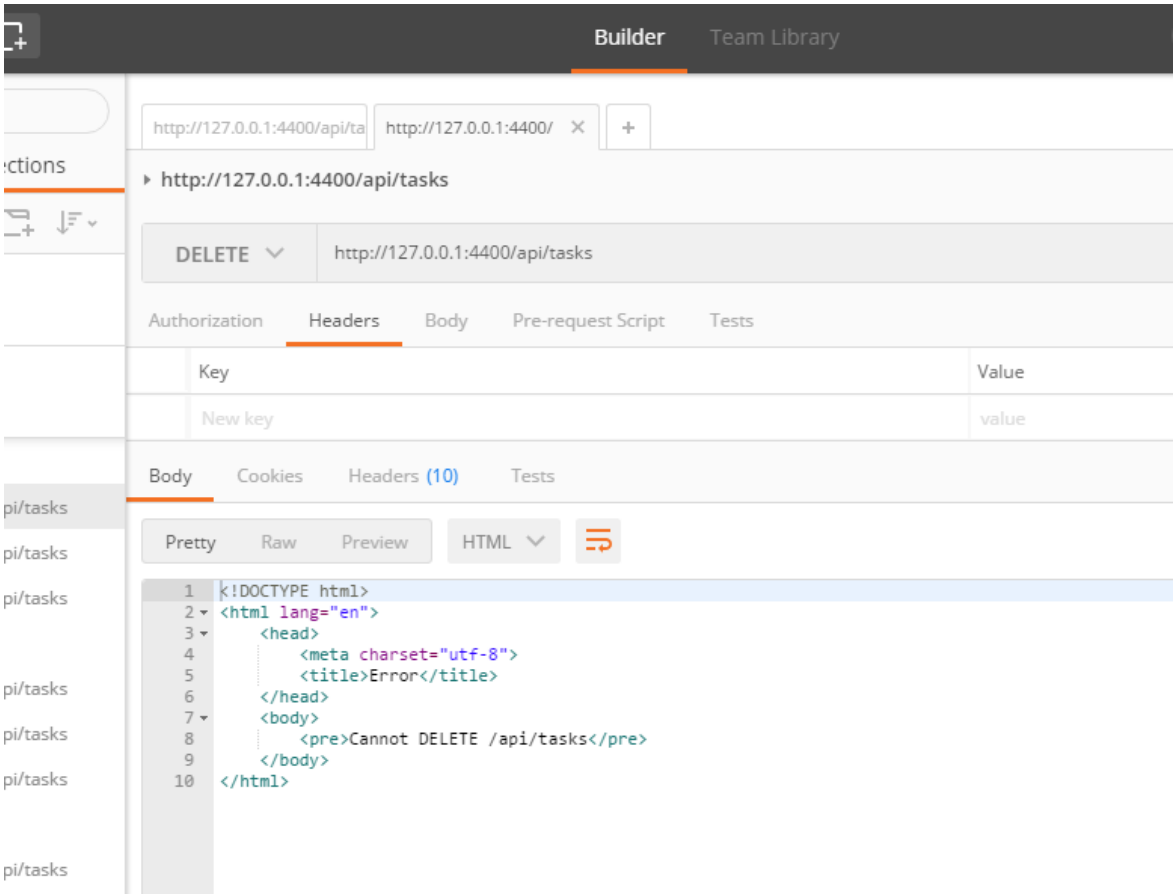
Prvým krokom testovania je vytvorenie si vlastnej kolekcie. Po vytvorení kolekcie je v rámci nej možné pridávať ďalšie súbory. Umožňuje to oddeliť od seba rôzne druhy volaní.

Na obrázku je vidieť vytvorenú kolekciu todoApp API. Pod ňou sú vytvorené súbory /task, /tasks/:task_id a /tasks/edit/:task_id. Z toho je zrejmé, že pod každým súborom sa skrýva testovanie jednotlivých definovaných endpointov.



Obrázok 25: Štruktúra testovacích súborov v aplikácii Postman

Prvý testom bolo testovanie endopointov pre implementované metódy. Vstupným predpokladom pre testy bolo, že definované metódy skončia so statusom 200 OK, akékoľvek iná, ako definovaná, dostane error, resp. 404 Not Found. Príkladom je DELETE pre endpoint /tasks. V špecifikácii má zadefinované len metódy POST a GET, takže DELETE by mal skončiť s chybou, čo sa aj stalo, viď obrázok nižšie.



Obrázok 26: Chybné volanie operácie DELETE pre endpoint /tasks

Okrem tohto rýchleho testu je možné testovanie pomocou javascriptových skriptov, ktoré sa píšu a spúšťajú priamo v termináli aplikácie. Niektoré jednoduché ponúka aj sám Postman.

Príklad testovania edpointu /task pomocou skriptov

Zoznam testovacích scenárov:

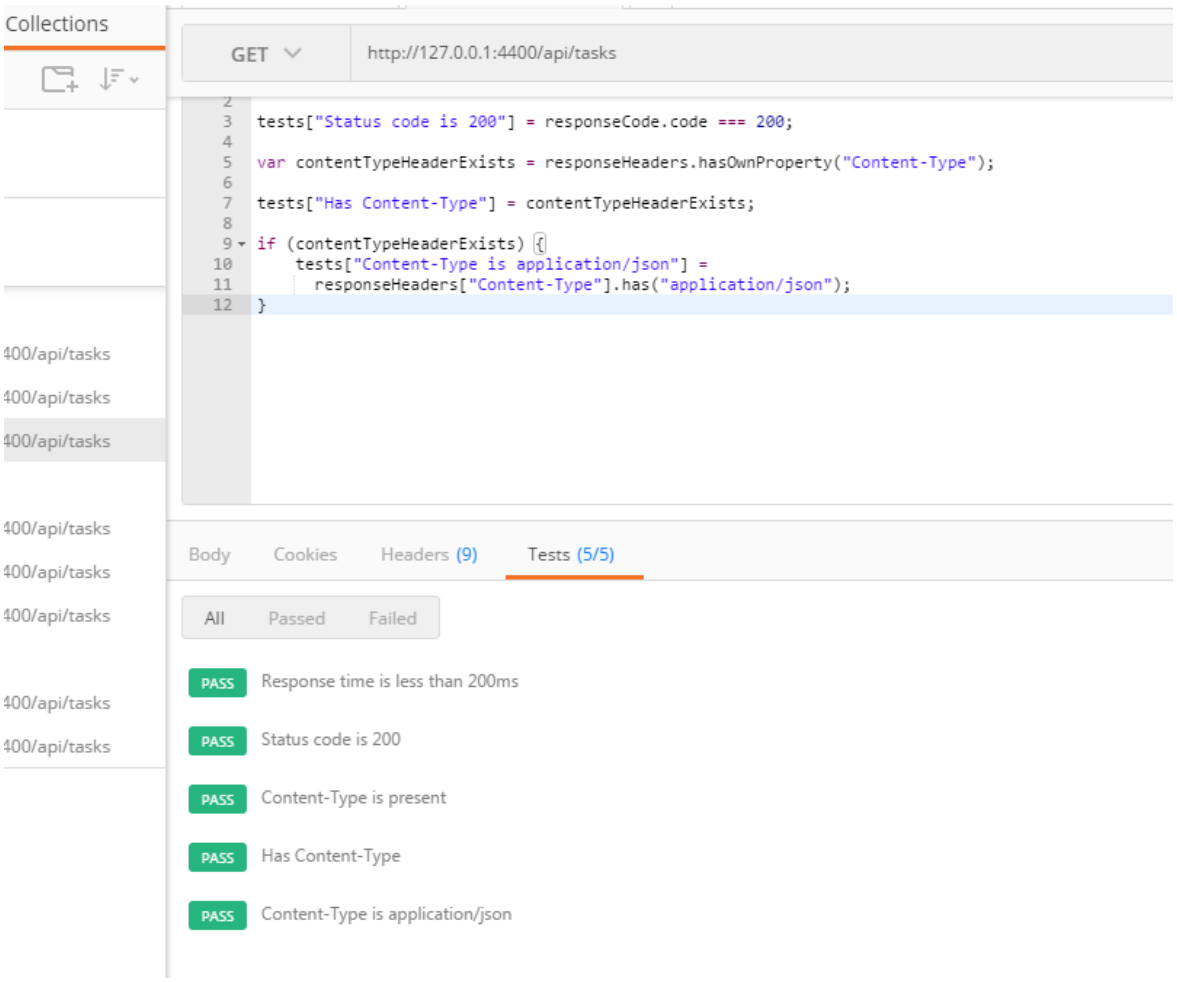
1. Chcem vedieť, či čas odpovede na moju požiadavku trvá menej, ako 200 sekúnd.
2. Chcem vedieť, či volanie skončí so statusom 200 OK.
3. Chcem vedieť, či hlavička Content-Type existuje.

4. Chcem vedieť, či je hodnota hlavičky Content-Type, ak existuje, rovná "application/json".

Testovací kód:

```
tests["Response time is less than 200ms"] = responseTime < 200;

tests["Status code is 200"] = responseCode.code === 200;

var contentTypeHeaderExists = responseHeaders.hasOwnProperty("Content-Type");

tests["Has Content-Type"] = contentTypeHeaderExists;

if (contentTypeHeaderExists) {
    tests["Content-Type is application/json"] =
      responseHeaders["Content-Type"].has("application/json");
}
```

Výsledok uvedených testov je možné vidieť na obrázku nižšie.



Obrázok 27: Výsledok testovania v aplikácii Postman pomocou skriptov

ZÁVĚR

V diplomovej práci som sa venovala problematike návrhu RESTového API pre prístup do databázy. Uvedenú problematiku som spracovala v dvoch častiach, Teoretickej a Praktickej, a v šiestich kapitolách. V prvej kapitole Základné pojmy som pre lepšie pochopenie RESTových služieb stručne vysvetlila niektoré základné pojmy, ktoré by mali objasniť miesto týchto služieb v širšej tématike webových aplikácií. Jednalo sa predovšetkým o vysvetlenie pojmu webová aplikácia, stručné zhrnutie webových komponentov a ich funkcionalít, ktoré vstupujú medzi odosielateľa a adresáta http(s) správy, a ktoré ovplyvňujú bezpečnosť a rýchlosť komunikácie, úvod do HTTP(s) protokolu (problematike komunikácie prostredníctvom tohto protokolu som sa venovala podrobnejšie v kapitole 2. RESTové služby. Jedná sa ale základný pojem, preto som ho stručne uviedla aj medzi základnými pojmami v širšom kontexte webových aplikácií) a objasnenie základných princípov a histórie webových služieb. V druhej kapitole som sa už podrobne venovala princípom RESTovej architektúry, popisom nefunkčných požiadaviek na RESTové rozhranie a ďalším témam, súvisiacim s RESTovou architektúrou. Tretia kapitola obsahuje podrobnejší opis najčastejšie využívaných frameworkov, používaných na vývoj API. REST architektúra sa neviaže na žiadny programovancí jazyk. Na jej implementáciu môžeme použiť ktorýkoľvek z dostupných frameworkov, napríklad zo sveta Javy, Node.js, Pythonu, PHP alebo Ruby. Zoznam frameworkov, podrobených analýze, bol zostavený na základe obľúbenosti na diskusných fórach, ako je stackoverflow.com alebo quora.com a tiež podľa prieskumu, ktorý realizoval optimalbi na svojom blogu v rámci série What's the best restful web api framework.

Štvrtá kapitola Analýza požiadaviek už patrí do Praktickej časti. Opísala som v nej funkčné a nefunkčné požiadavky na aplikáciu. Cieľom analýzy požiadaviek bolo zistiť, aký problém treba riešiť a aké sú požiadavky na aplikáciu. Na základe týchto vstupných údajov som sa potom rozhodla, aké technológie budú vhodné na realizáciu. V mojom prípade bolo požiadavkou vytvoriť RESTové API pre prístup do databázy a následne jednoduchú aplikáciu, na ktorej bude možné prezentovať funkčnosť tohto API. V piatej kapitole som predstavila technológie, s pomocou ktorých som naprogramovala aplikáciu s funkčným API rozhraním. Šiesta, posledná kapitola, opisuje samotnú realizáciu a ukážky kódov.

Výstupom je funkčná aplikácia, ktorá tvorí prílohu diplomovej práce.

Stručný popis vývoja aplikácie

Pre vývoj aplikácie som použila kombináciu nástrojov, známych pod skratkou MEAN. Skrýva sa po ňou mongoDB, Express, Angular a Node.js. Práca s uvedenými technológiami je jednoduchá. Keďže sa jedná o obľúbené nástroje, ku každej z nich je k dispozícii množstvo prehľadnej dokumentácie, postupov a návodov na riešenie problémov.

Keďže som implementovala iba jednoduché operácie CRUD-u, nenarazila som počas vývoja aplikácie na žiaden výraznejší problém.

Kontrola stanovených cieľov

Cieľom diplomovej práce, ktorý som si stanovila na začiatku, bolo:

> navrhnúť RESTové API pre prístup do databázy. Pred samotným návrhom preštudovať teoretické informácií o RESTovom rozhraní, jeho bezpečnosti a s ním súvisiacom HTTP protokole. Na základe týchto informácií vytvoriť požadované API a jeho funkčnosť následne demonštrovať v rámci jednoduchej aplikácie.
>
> Okrem návrhu samotného API zostaviť zoznam funkčných a nefunkčných požiadaviek, kladených na aplikáciu. Následne zvoliť vhodné softvérové prostriedky ako pre frontend, tak pre backend, navrhnúť databázu, wirefram pre fronted a popísať rozhrania pre backendové služby.

Stanovený cieľ som splnila. Popis teórie REST API, aj návrh a spôsob fungovania aplikácie je obsahom tejto diplomovej práce. Zdrojový kód s navrhnutým API je súčasťou jej prílohy.

Návrhy na zlepšenie

Práca implementuje len jednoduché CRUD operácie. Tiež obsahuje len základné funkcionality. Do budúcnosti by bolo možné rozšíriť aplikáciu o prihlasovanie, validáciu polí, rôzne druhy filtrovania a pod.

SEZNAM POUŽITÉ LITERATURY

[1] https://www.w3.org/TR/wsdl [cit. 2017-04-16].

[2] HANÁČEK, Petr, STAUDEK, Jan. *Certifikační infrastruktury veřejných klíču, PKI*. Dostupné na:

http://www.fi.muni.cz/usr/staudek/vyuka/security/stud_lit/D01_C.pdf

[3] SCHMIDL, Tomáš. *Návrh a implementace RESTových rozhraní.* Brno, 2016. Bakalářská práce. Masarykova univezita, Fakulta informatiky. Vedoucí práce Petr Sojka. Dostupné tiež na: http://is.muni.cz/th/410087/fi_b/navrh-a-implementace-restovych-rozhrani.pdf

[4] WEBBER, Jim, PARASTATIDIS, Savas a ROBINSON, Ian. *REST in Practice*. O'Reilly Media, Inc., 2010. ISBN: 978-0-596-80582-1.

[5] FIELDING, Roy Thomas. Architectural styles and the design of network-based software architectures. Irvine, 2000. ISBN 0-599-87118-0. Disertačná práca. University of California, Irvine. Vedúci práce Richard Taylor. Dustupné tiež na: http://www.ics.uci.edu/~fielding/pubs/dissertation/net_app_arch.htm

[6] RICHARDSON, Leonard, AMUNDSEN, Michael. *RESTfulWeb APIs*. O'Reilly Media, Inc., 2013. ISBN 9781449358068.

[7] ERL, Thomas: *SOA Servisně orientovaná architektura*. Computer Press, 2009. ISBN: 9788025118863.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

FE	Frontend
HTTP	Hypertext Transfer Protocol.
HTTPS	Hyper Text Transfer Protocol Secure.
IDS	Intrusion detection system.
IPS	Intrusion Prevention System.
J2EE	Java 2 Platform, Enterprise Edition.
JSON	JavaScript Object Notation.
JSON	JavaScript Object Notation.
ODM	Object Data Mapping.
REST	Representational state transfer.
RPC	Remote Procedure Call.
SOA	Service-oriented architecture.
SOAP	Simple Object Access Protocol.
UC	Use Case.
UDDI	Universal Description, Discovery, and Integration.
URI	Uniform Resource Identifier.
URN	Uniform Resource Name.
WSDL	Web Service Definition Language.
XML	eXtensible Markup Language.
XSD	XML Schema Definition.

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: OBSAH PRILOŽENÉHO CD

Priložené CD obsahuje:

- fulltext.pdf – diplomová práca
- todoApp – zrojové kódy aplikácie
- README.txt – pokyny pre vývojové prostredie a spustenie aplikácie